7ª edición

Organización y arquitectura de computadores

www.librosite.net/stallings1



William Stallings

PEARSON
Prentice Hall

Organización y arquitectura de computadores

Organización y arquitectura de computadores

Séptima Edición

WILLIAM STALLINGS

Traducción
Antonio Cañas Vargas
Beatriz Prieto Campos
Francisco José Pelayo Valle
Julio Ortega Lopera
Departamento de Arquitectura y Tecnología de Computadores
Universidad de Granada

Coordinación y revisión técnica
Alberto Prieto Espinosa
Departamento de Arquitectura y Tecnología de Computadores
Universidad de Granada



Madrid • México • Santafé de Bogotá • Buenos Aires • Caracas • Lima • Montevideo • San Juan •
San José • Santiago • São Paulo • White Plains

Datos de catalogación bibliográfica

ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORES
WILLIAM STALLINGS

PEARSON EDUCACIÓN, S.A., Madrid, 2005

ISBN 10: 84-8966-082-4
ISBN 13: 978-84-8966-082-3
Materia: Informática. 0004.4

Formato: 195 × 250 mm Páginas: 840

C/ Ribera del Loira, 28
28042 Madrid (España)

ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORES
WILLIAM STALLINGS

Última reimpresión: 2007

ISBN: 978-84-8966-082-3
Depósito Legal: M-15.392-2007

PEARSON-PRENTICE HALL es un sello editorial autorizado de PEARSON EDUCACIÓN, S.A.

Authorized translation from the English language edition, entitled COMPUTER ORGANIZATION AND ARCHITECTURE: DESIGNING FOR PERFOMANCE, 7th Edition, by STALLINGS, WILLIAM, published by Pearson Education, Inc, publishing as Prentice Hall, Copyright © 2006.
ISBN: 0-13-146592-9

Equipo editorial:
Editor: Miguel Martín-Romo
Técnico editorial: Marta Caicoya
Equipo de producción:
Director: José Antonio Clares
Técnico: José Antonio Hernán
Diseño de cubierta: Equipo de diseño de Pearson Educación, S.A.

Focomposición: JOSUR, TRATAMIENTO DE TEXTOS, S.L.

Impreso en México. *Printed in Mexico.*

Contenido

CAPÍTULO 0

Guía del lector

Este libro, junto con su sitio web, cubre mucha materia. El presente capítulo da al lector un resumen.

0.1. ESQUEMA DEL LIBRO

El libro está organizado en cinco partes:

Parte uno: es una descripción de la organización y arquitectura de los computadores y muestra cómo ha evolucionado su diseño.

Parte dos: se examinan la mayoría de los componentes de un computador y sus interconexiones, tanto internas como externas. En esta parte también se incluye una descripción detallada de la memoria interna y externa y de las E/S. Finalmente, se examina la relación entre la arquitectura de un computador y el sistema operativo de esa arquitectura.

Parte tres: se examinan la arquitectura interna y la organización del procesador. Comienza con una descripción extensa de la aritmética del computador. Luego se ve la arquitectura del conjunto de instrucciones. El resto de esta parte se ocupa de la estructura y funcionamiento del procesador, incluyendo una descripción de las arquitecturas RISC y superescalares, así como una visión detallada de la arquitectura IA-64.

Parte cuatro: se describe la estructura interna de la unidad de control del procesador y el uso de la microprogramación.

Parte cinco: se ocupa de la organización paralela, incluyendo multiprocesamiento simétrico y *clusters*.

Al principio de cada capítulo hay un resumen más detallado de cada parte.

La finalidad de este texto es que el lector aprenda los principios de diseño e implementación de la organización y arquitectura de los computadores actuales. Por consiguiente, un tratamiento puramente conceptual o teórico sería inadecuado. Este libro utiliza ejemplos de muchas máquinas diferentes para clarificar y reforzar los conceptos presentados. Muchos, pero no todos, de los ejemplos se han ideado a partir de dos familias de computadores: Intel Pentium 4 y PowerPC IBM/Freescale. Estos dos sistemas juntos abarcan la mayoría de las tendencias en diseño de los computadores de hoy en día. El Pentium 4 es esencialmente un computador con un conjunto complejo de instrucciones (CISC, *Complex Instruction Set Computer*) con características RISC, mientras que el PowerPC es esencialmente un computador con un conjunto reducido de instrucciones (RISC, *Reduced Instruction Set Computer*). Ambos sistemas utilizan principios de diseño superescalar y ambos soportan configuraciones multiprocesador.

0.2. INTERNET Y RECURSOS WEB

Hay una serie de recursos disponibles en Internet y en la web para complementar este libro y para ayudar a progresar en este campo.

Se ha creado una página web para este libro en WilliamStallings.com/COA/COA7e.html. Ver las dos páginas iniciales de este libro para una descripción detallada de este sitio.

Se mantendrá una lista de erratas del libro en el sitio web y se actualizará según las necesidades. Por favor, envíe por e-mail los errores que encuentre. Las hojas de erratas de mis otros libros están en WilliamStallings.com.

También mantengo el sitio de recursos del estudiante de informática, en WilliamStallings.com/StudentSupport.html; la finalidad de este sitio es proporcionar documentos, información y enlaces útiles a estudiantes y profesionales de la informática. Los enlaces están organizados en cuatro categorías:

- **Matemáticas**: incluye un repaso de matemáticas básicas, análisis básico de teoría de colas, sistemas de numeración básicos, y enlaces útiles a sitios web sobre matemáticas.
- **Resolución**: consejos y guías para resolver problemas propuestos, informes técnicos escritos y para preparar presentaciones técnicas.
- **Recursos de investigación**: enlaces a numerosos artículos, informes técnicos y bibliografía.
- **Varios**: una serie de documentos y enlaces útiles.

OTROS SITIOS WEB

Hay varios sitios web con información relacionada con los temas que se tratan en este libro. En los siguientes capítulos, en la sección de «Lecturas recomendadas y sitios web», se pueden encontrar enlaces a sitios web específicos. Debido a que las URL de los sitios Web tienden a cambiar con frecuencia, no están incluidas en este libro. Los enlaces de todos los sitios Web listados en el libro se pueden encontrar en el sitio web de este libro. Se han añadido otros enlaces cuando se ha creído oportuno.

A continuación se listan sitios web de interés general relacionados con la organización y arquitectura de computadores:

- **Página principal de WWW Computer Architecture:** índice exhaustivo de información relacionada con investigadores de arquitectura de computadores, incluyendo grupos de arquitectura y proyectos, organizaciones técnicas, literatura, empleo e información comercial.
- **CPU Info Center:** información sobre procesadores específicos, incluyendo artículos técnicos, información sobre productos y últimas novedades.
- **Emporio del procesador**: una colección interesante y útil de información.
- **ACM Special Interest Group on Computer Architecture:** información sobre actividades y publicaciones SIGARCH.
- **Comité técnico del IEEE en arquitectura de computadores:** copias del boletín de noticias de TCAA.

GRUPOS DE NOTICIAS USENET

Una serie de grupos de noticias USENET se dedica a algunos aspectos de la arquitectura y organización de computadores. Como con casi todos los grupos USENET, hay una relación señal-ruido alta, pero merece la pena ver experimentalmente si alguno satisface sus necesidades. Los más importantes son los siguientes:

- **Comp.arch:** un grupo de noticias general para hablar sobre arquitectura de computadores.
- **Comp.arch.arithmetic**: trata sobre algoritmos aritméticos de computadores y estándares.
- **Comp.arch.storage:** discusiones desde productos hasta tecnología sobre cuestiones de uso en la práctica.
- **Comp.parallel**: trata sobre computadores paralelos y sus aplicaciones.

PARTE 1

VISIÓN GENERAL

CUESTIONES A TRATAR EN LA PRIMERA PARTE

El objetivo de la primera parte es proporcionar una base y un contexto para el resto del libro. Se presentan los conceptos fundamentales sobre arquitectura y organización de computadores.

ESQUEMA DE LA PRIMERA PARTE

CAPÍTULO 1. INTRODUCCIÓN

El Capítulo 1 introduce el concepto de computador como sistema jerárquico. Un computador puede ser visto como una estructura de componentes y su funcionamiento puede ser descrito en términos del funcionamiento colectivo de sus componentes cooperantes. Cada componente puede ser descrito, a su vez, según su estructura interna y funcionamiento. Se introducen los niveles principales de esta visión jerárquica. El resto del libro está organizado, comenzando por el nivel superior y bajando hasta los inferiores, según estos niveles.

CAPÍTULO 2. FUNDAMENTO Y EVOLUCIÓN DE LOS COMPUTADORES

El Capítulo 2 tiene dos finalidades. Primero, hablar de la historia de la tecnología de computadores es una forma sencilla e interesante de introducir conceptos básicos sobre organización y arquitectura de computadores. El capítulo también trata las tendencias en tecnología que son el fundamento del diseño de computadores y que han previsto distintas técnicas y estrategias que se han usado para conseguir un funcionamiento equilibrado y eficiente.

CAPÍTULO 1

Introducción

Este libro trata sobre la estructura y funcionamiento de los computadores. Su objetivo es presentar, tan clara y completamente como sea posible, la naturaleza y las características de los computadores de hoy día. Este objetivo es todo un reto por dos razones.

Primero, hay una gran variedad de sistemas que pueden recibir correctamente el nombre de *computador*, desde micropocesadores de un solo chip, que cuestan unos pocos dólares, a supercomputadores que cuestan decenas de millones de dólares. Esta variedad es patente no solo en costes sino en tamaño, prestaciones y aplicaciones. Segundo, el rápido ritmo de cambio que ha caracterizado siempre a la tecnología de computadores continúa sin pausa. Estos cambios cubren todos los aspectos de la tecnología de computadores. Desde la tecnología subyacente de circuitos integrados, usados para construir componentes de computadores, hasta el creciente uso de conceptos de organización paralela para combinar esos componentes.

A pesar de la variedad y el ritmo de cambio en el campo de los computadores, se aplican sistemáticamente ciertos conceptos fundamentales. La aplicación de estos conceptos depende del desarrollo actual de la tecnología y de los objetivos en precio-aplicación del diseñador. La intención de este libro es ofrecer un concienzudo análisis de los fundamentos de la arquitectura y organización de los computadores y relacionar estos con materias de diseño actuales. Este capítulo introduce la aproximación descriptiva que se va a considerar.

1.1. ORGANIZACIÓN Y ARQUITECTURA

Cuando se describe un computador, frecuentemente se distingue entre *arquitectura* y *organización*. Aunque es difícil dar una definición precisa para estos términos, existe un consenso sobre las áreas generales cubiertas por cada uno de ellos (por ejemplo, *veáse* [VRAN80], [SIEW82], y [BELL78a]).

La arquitectura de computadores se refiere a los atributos de un sistema que son visibles a un programador, o para decirlo de otra manera, aquellos atributos que tienen un impacto directo en la ejecución lógica de un programa. La organización de computadores se refiere a las unidades funcionales y sus interconexiones, que dan lugar a especificaciones arquitectónicas. Entre los ejemplos de atributos arquitectónicos se encuentran el conjunto de instrucciones, el número de bits usados para representar varios tipos de datos (por ejemplo, números, caracteres), mecanismos de E/S y técnicas para direccionamiento de memoria. Entre los atributos de organización se incluyen aquellos detalles de hardware transparentes al programador, tales como señales de control, interfaces entre el computador y los periféricos y la tecnología de memoria usada.

Para poner un ejemplo, una cuestión de diseño arquitectónico es si el computador tendrá la instrucción de multiplicar. Una cuestión de organización es si esa instrucción será implementada por una unidad especializada en multiplicar o por un mecanismo que haga un uso iterativo de la unidad de suma del sistema. La decisión de organización puede estar basada en la frecuencia prevista del uso de la instrucción de multiplicar la velocidad relativa de las dos aproximaciones, y el coste y el tamaño físico de una unidad especializada en multiplicar.

Históricamente, y aún hoy día, la distinción entre arquitectura y organización ha sido importante. Muchos fabricantes de computadores ofrecen una familia de modelos, todos con la misma arquitectura pero con diferencias en la organización. Consecuentemente los diferentes modelos de la familia tienen precios y prestaciones distintas. Más aún, una arquitectura puede sobrevivir muchos años, pero su organización cambia con la evolución de tecnología. Un ejemplo destacado de ambos fenómenos

es la arquitectura IBM Sistema/370. Esta arquitectura apareció por primera vez en 1970 e incluía varios modelos. Un cliente con necesidades modestas podía comprar un modelo más barato y lento, y, si la demanda se incrementaba, cambiarse más tarde a un modelo más caro y rápido sin tener que abandonar el software que ya había sido desarrollado. A través de los años IBM ha introducido muchos modelos nuevos con tecnología mejorada para reemplazar a modelos más viejos, ofreciendo al consumidor mayor velocidad, precios más bajos o ambos a la vez. Estos modelos más nuevos conservaban la misma arquitectura para proteger así la inversión en software del consumidor. Podemos destacar que la arquitectura del Sistema/370 con unas pocas mejoras ha sobrevivido hasta hoy día como la arquitectura de la línea de grandes productos de computación IBM.

En una clase de sistemas, llamados microcomputadores, la relación entre arquitectura y organización es muy estrecha. Los cambios en la tecnología no solo influyen en la organización, sino que también dan lugar a la introducción de arquitecturas más ricas y potentes. Generalmente hay menos requisitos de compatibilidad generación a generación para estas pequeñas máquinas. Así, hay más interacción entre las decisiones de diseño arquitectónicas y de organización. Un ejemplo interesante de esto son los computadores de repertorio reducido de instrucciones (RISC, *Reduced Instruction Set Computer*), que veremos en el Capítulo 13.

En este libro se examina tanto la organización como la arquitectura de un computador. Se da, quizá, más énfasis a la parte de organización. Sin embargo, como la organización de un computador debe ser diseñada para implementar la especificación de una arquitectura particular, un estudio exhaustivo de la organización requiere también un análisis detallado de la arquitectura.

1.2. ESTRUCTURA Y FUNCIONAMIENTO

Un computador es un sistema complejo; los computadores de hoy en día contienen millones de componentes electrónicos básicos. ¿Cómo podríamos describirlos claramente? La clave está en reconocer la naturaleza jerárquica de la mayoría de los sistemas complejos, incluyendo el computador [SIMO69]. Un sistema jerárquico es un conjunto de subsistemas interrelacionados cada uno de los cuales, a su vez, se organiza en una estructura jerárquica hasta que se alcanza el nivel más bajo del subsistema elemental.

La naturaleza jerárquica de los sistemas complejos es esencial tanto para su diseño como para su descripción. El diseñador necesita tratar solamente con un nivel particular del sistema a la vez. En cada nivel el sistema consta de un conjunto de componentes y sus interrelaciones. El comportamiento en cada nivel depende solo de una caracterización abstracta y simplificada del sistema que hay en el siguiente nivel más bajo. De cada nivel al diseñador le importan la estructura y el funcionamiento:

- **Estructura***:* el modo en que los componentes están interrelacionados.
- **Funcionamiento***:* la operación de cada componente individual como parte de la estructura.

En términos de descripción tenemos dos opciones: empezar por lo más bajo y construir una descripción completa, o comenzar con una visión desde arriba y descomponer el sistema en sus subpartes. La experiencia a partir de muchos campos nos ha enseñado que la descripción de arriba abajo (*top-down*) es la más clara y efectiva [WEIN75].

El enfoque seguido en este libro considera este punto de vista. El computador será descrito de arriba abajo. Comenzamos con los componentes principales del sistema describiendo su estructura y

funcionamiento, y seguimos sucesivamente hacia capas más bajas de la jerarquía. Lo que queda de esta sección ofrece una breve visión global de este plan de ataque.

FUNCIONAMIENTO

Tanto la estructura como el funcionamiento de un computador son en esencia sencillos. La Figura 1.1 señala las funciones básicas que un computador puede llevar a cabo. En términos generales hay solo cuatro:

- Procesamiento de datos
- Almacenamiento de datos
- Transferencia de datos
- Control

El computador, por supuesto, tiene que ser capaz de **procesar datos**. Los datos pueden adoptar una gran variedad de formas, y el rango de los requisitos de procesado es amplio. Sin embargo, veremos que hay solo unos pocos métodos o tipos fundamentales de procesado de datos.

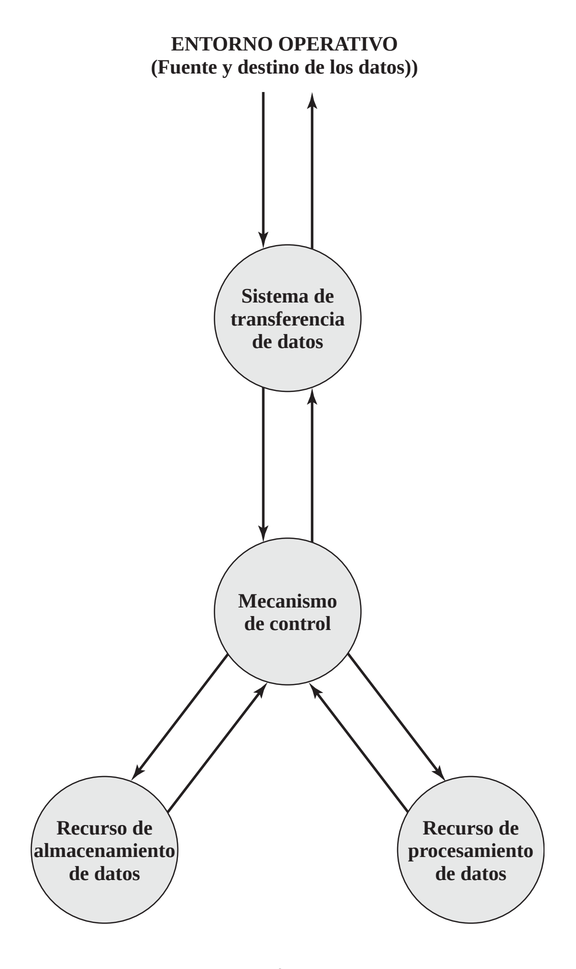


Figura 1.1. Una visión funcional de un computador.

También es esencial que un computador **almacene datos**. Incluso si el computador está procesando datos al vuelo (es decir, los datos se introducen, se procesan, y los resultados se obtienen inmediatamente), el computador tiene que guardar temporalmente al menos aquellos datos con los que está trabajando en un momento dado. Así hay al menos una función de almacenamiento de datos a corto plazo. Con igual importancia el computador lleva a cabo una fución de almacenamiento de datos a largo plazo. El computador almacena ficheros de datos para que se recuperen y actualicen en un futuro.

El computador tiene que ser capaz de **transferir datos** entre él mismo y el mundo exterior. El entorno de operación del computador se compone de dispositivos que sirven bien como fuente o bien como destino de datos. Cuando se reciben o se llevan datos a un dispositivo que está directamente conectado con el computador, el proceso se conoce como *entrada-salida* (E/S), y este dispositivo recibe el nombre de *periférico*. El proceso de transferir datos a largas distancias, desde o hacia un dispositivo remoto, recibe el nombre de *comunicación de datos*.

Finalmente, debe haber un **control** de estas tres funciones. Este control es ejercido por el(los) ente(s) que proporciona(n) al computador instrucciones. Dentro del computador, una unidad de control gestiona los recursos del computador y dirige las prestaciones de sus partes funcionales en respuesta a estas instrucciones.

A este nivel general de discusión, el número de operaciones posibles que pueden ser realizadas es pequeño. La Figura 1.2 muestra los cuatro posibles tipos de operaciones. El computador puede funcionar como un dispositivo de transferencia de datos (Figura 1.2a), simplemente transfiriendo datos de un periférico o línea de comunicaciones a otro. También puede funcionar como un dispositivo de almacenamiento de datos (Figura 1.2b), con datos transferidos desde un entorno externo al almacén de datos del computador (leer) y viceversa (escribir). Los dos diagramas siguientes muestran operaciones que implican procesamiento de datos, en datos, o bien almacenados (Figura 1.2c) o en tránsito entre el almacén y el entorno externo (Figura 1.2d).

La exposición precedente puede parecer absurdamente generalizada. Es posible, incluso en el nivel más alto de la estructura de un computador, diferenciar varias funciones, pero citando [SIEW82]:

> «Hay, sorprendentemente, muy pocas formas de estructuras de computadores que se ajusten a la función que va a ser llevada a cabo. En la raíz de esto subyace el problema de la naturaleza de uso general de los computadores, en la cual toda la especialización funcional se tiene cuando se programa y no cuando se diseña».

ESTRUCTURA

La Figura 1.3 es la representación más sencilla posible de un computador. El computador es una entidad que interactúa de alguna manera con su entorno externo. En general, todas sus conexiones con el entorno externo pueden ser clasificadas como dispositivos periféricos o líneas de comunicación. Diremos algo más adelante sobre ambos tipos de conexiones.

Pero tiene más interés en este libro la estructura interna del computador mismo, que mostramos, en su nivel más alto, en la Figura 1.4. Hay cuatro componentes estructurales principales:

- **Unidad Central de Procesamiento (CPU, *Central Processing Unit*)**: controla el funcionamiento del computador y lleva a cabo sus funciones de procesamiento de datos. Frecuentemente se le llama simplemente **procesador**.

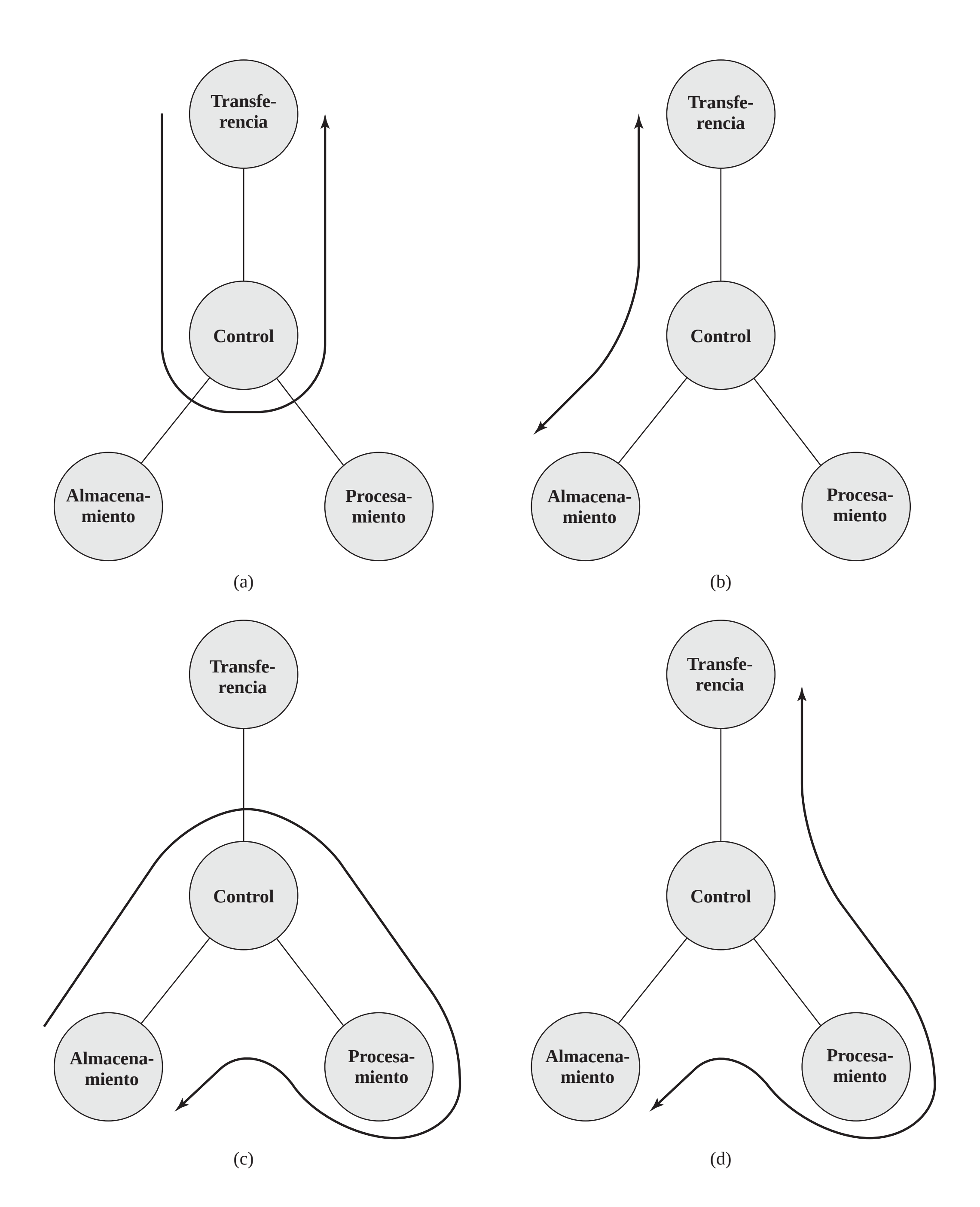


Figura 1.2. Posibles operaciones de un computador.

- **Memoria principal:** almacena datos.
- **E/S:** transfiere datos entre el computador y el entorno externo.
- **Sistema de interconexión:** es un mecanismo que proporciona la comunicación entre la CPU, la memoria principal y la E/S.

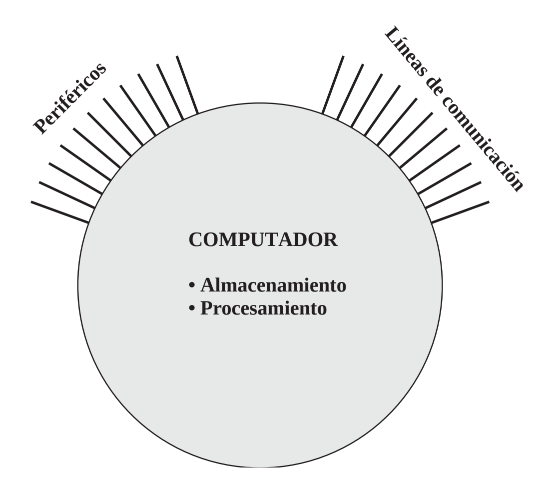


Figura 1.3. El computador.

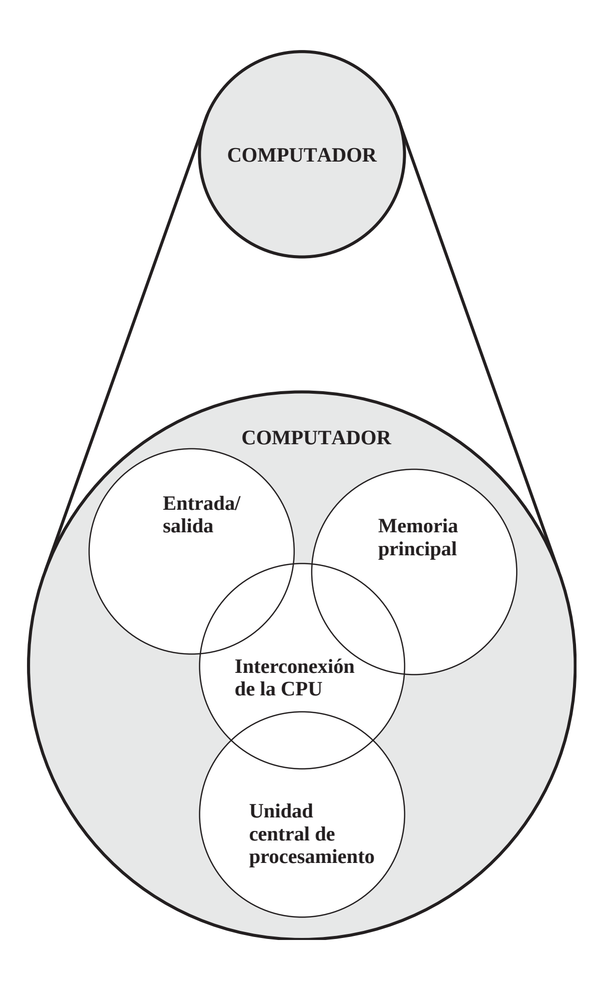


Figura 1.4. El computador: estructura del nivel superior.

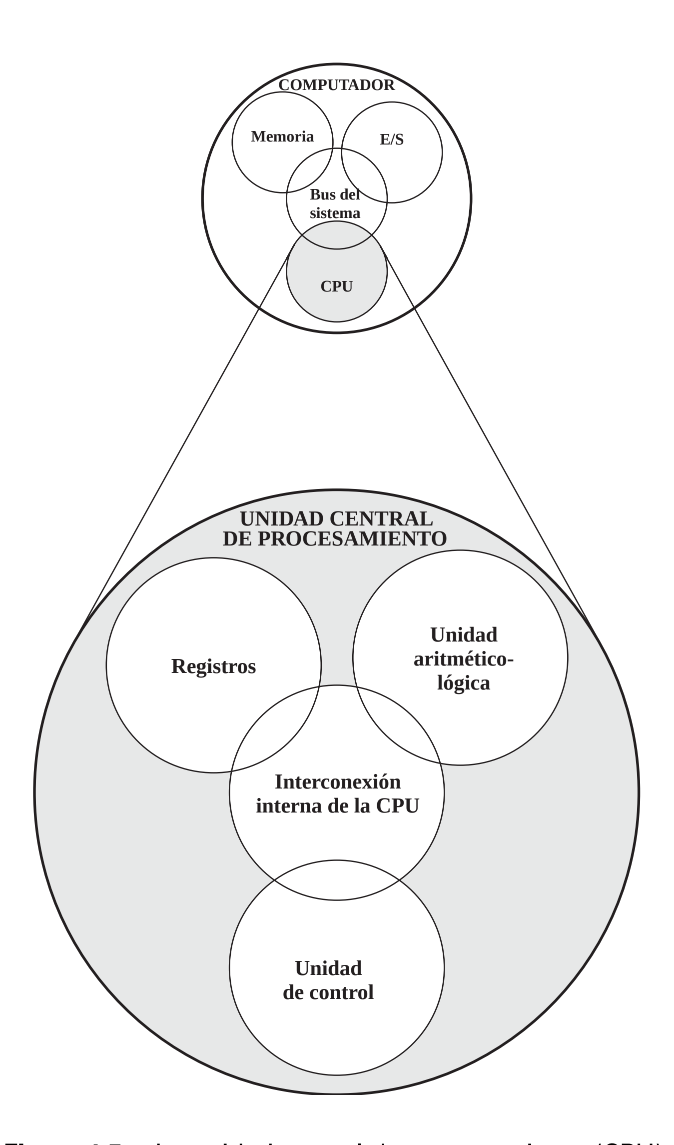


Figura 1.5. La unidad central de procesamiento (CPU).

Puede que haya uno o más de cada uno de estos componentes. Tradicionalmente ha habido solo una CPU. En los últimos años ha habido un uso creciente de varios procesadores en un solo sistema. Surgen algunas cuestiones relativas a multiprocesadores y se discuten conforme el texto avanza; la Parte Cinco se centra en tales sistemas.

Cada uno de estos componentes será examinado con cierto detalle en la Parte Dos. Sin embargo, para nuestros objetivos, el componente más interesante y de algún modo el más complejo es la CPU; su estructura se muestra en la Figura 1.5. Sus principales componentes estructurales son:

- **Unidad de control:** controla el funcionamiento de la CPU y por tanto del computador.
- **Unidad aritmético-lógica (ALU, *Arithmetic Logic Unit*):** lleva a cabo las funciones de procesamiento de datos del computador.

- **Registros:** proporcionan almacenamiento interno a la CPU.
- **Interconexiones CPU:** son mecanismos que proporcionan comunicación entre la unidad de control, la ALU y los registros.

Cada uno de estos componentes será analizado con detalle en la Parte Tres, donde veremos que la complejidad aumenta con el uso de técnicas de organización paralelas y de segmentación de cauce. Finalmente, hay varias aproximaciones para la implementación de la unidad de control; una de las aproximaciones más comunes es la implementación *microprogramada*. Básicamente, una unidad de control microprogramada actúa ejecutando microinstrucciones que definen la funcionalidad de la unidad de control. Con esta aproximación, la estructura de la unidad de control puede ser como la mostrada en la Figura 1.6. Esta estructura será examinada en la Parte Cuatro.

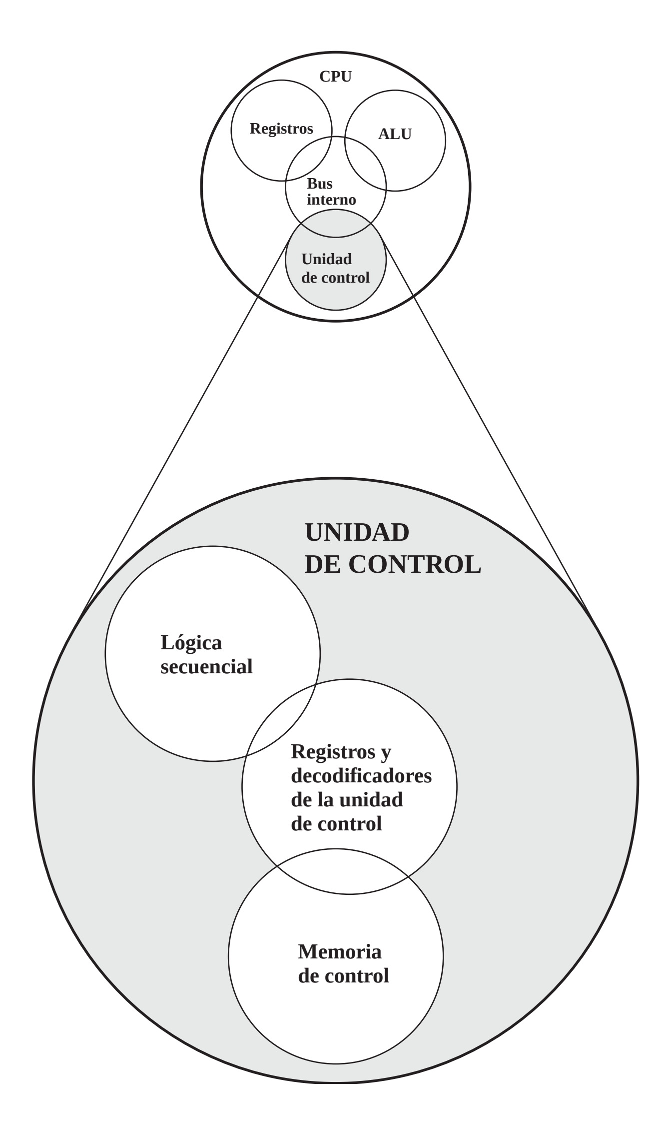


Figura 1.6. La unidad de control.

1.3. ¿POR QUÉ ESTUDIAR LA ORGANIZACIÓN Y ARQUITECTURA DE LOS COMPUTADORES?

El «IEEE/ACM Computer Curricula 2001» [JTF01], preparado por la *Joint Task Force* de currículo de computadores de la Sociedad de Computadores IEEE (*Institute of Electrical and Electronics Engineers*) y la ACM (*Association for Computing Machinery*), citan la arquitectura de computadores como uno de los temas troncales que debe estar en todos los currículos de todos los estudiantes de licenciatura e ingeniería informática. El informe dice lo siguiente:

> «El computador está en el corazón de la informática. Sin él la mayoría de las asignaturas de informática serían hoy una rama de la matemática teórica. Para ser hoy un profesional en cualquier campo de la informática uno no debe ver al computador como una caja negra que ejecuta programas mágicamente. Todos los estudiantes de informática deben, en cierta medida, comprender y valorar los componentes funcionales de un computador, sus características, su funcionamiento y sus interacciones. También sus implicaciones prácticas. Los estudiantes necesitan comprender la arquitectura del computador para estructurar un programa de forma que este sea más eficiente en una máquina real. Seleccionando el sistema que se va a usar, debe ser capaz de comprender el compromiso entre varios componentes, como la velocidad del reloj de la CPU frente al tamaño de la memoria».

En [CLEM00] se dan los siguientes ejemplos como razones para estudiar arquitectura de computadores:

1. Supóngase que un licenciado trabaja en la industria y se le pide seleccionar el computador con la mejor relación calidad precio para utilizarlo en una gran empresa. Comprender las implicaciones de gastar más en distintas alternativas, como una caché grande o una velocidad de reloj mayor, es esencial para tomar esta decisión.
2. Hay muchos procesadores que no forman pare de equipos PC o servidores, pero sí en sistemas embebidos. Un diseñador debe ser capaz de programar un procesador en C que esté embebido en algún sistema en tiempo real o sistema complejo, como un controlador electrónico de un coche inteligente. Depurar el sistema puede requerir utilizar un analizador lógico que muestre la relación entre las peticiones de interrupción de los sensores del sistema y el código máquina.
3. Los conceptos utilizados en arquitectura de computadores tienen aplicación en otros cursos. En particular, la forma en la que el computador ofrece un soporte arquitectural a los lenguajes de programación y funciones en principio propias del sistema operativo, refuerza los conceptos de estas áreas.

Como se puede deducir del índice de este libro, la organización y arquitectura de computadores abarca un amplio rango de temas y conceptos. Una buena comprensión de estos conceptos será útil tanto en otras áreas de estudio como en un futuro trabajo después de licenciarse.

CAPÍTULO 2

Evolución y prestaciones de los computadores

PUNTOS CLAVE

- La evolución de los computadores se ha caracterizado por un incremento de la velocidad del procesador, una disminución del tamaño de los componentes, un aumento del tamaño de memoria, y un aumento de la capacidad de E/S y de la velocidad.
- Otro factor responsable del gran aumento de la velocidad del procesador es la disminución del tamaño de los componentes del microprocesador; esto reduce la distancia entre componentes, y, por tanto, aumenta la velocidad. Sin embargo, la verdadera ganancia en velocidad en los últimos años se debe a la organización del procesador, incluyendo un uso amplio de la segmentación de cauce, de las técnicas de ejecución paralela y del uso de técnicas de ejecución especulativa, que conducen a la ejecución tentativa de instrucciones futuras que se puedan necesitar. Todas estas técnicas se diseñan para mantener al procesador ocupado el mayor tiempo posible.
- Otro asunto crítico en el diseño de computadores es hacer un balance de las prestaciones de los distintos elementos, de forma que esta ganancia en prestaciones en un área no perjudique a otras áreas. En particular, la velocidad del procesador ha aumentado más rápidamente que el tiempo de acceso a memoria. Se han usado distintas técnicas para compensar este desacoplo, incluyendo memorias caché, caminos de datos más anchos de la memoria al procesador y más circuitos de memoria inteligentes.

Empezamos nuestro estudio de los computadores con una breve historia. Esta historia es interesante por sí misma y además sirve para proporcionar una visión general de la estructura y funcionamiento de los computadores. Luego se trata el tema de las prestaciones. La consideración de la necesidad de equilibrar los recursos de un computador nos da un contexto útil en todo el libro. Finalmente, veremos brevemente la evolución de dos sistemas que sirven como ejemplos clave en todo el libro: Pentium y PowerPC.

2.1. UNA BREVE HISTORIA DE LOS COMPUTADORES

LA PRIMER GENERACIÓN: LOS TUBOS DE VACÍO

ENIAC El ENIAC (*Electronic Numerical Integrator And Computer*), diseñado y construido bajo la supervisión de John Mauchly y John Presper Eckert en la Universidad de Pennsylvania, fue el primer computador electrónico de propósito general del mundo.

El proyecto fue una respuesta a necesidades militares durante la Segunda Guerra Mundial. El BRL (*Ballistcs Research Laboratory*, Laboratorio de Investigación de Balística) del Ejército, una agencia responsable del desarrollo de tablas de tiro y de trayectoria para nuevas armas, tenía dificultades para elaborarlas con exactitud y dentro de un plazo de tiempo razonable. Sin estas tablas de tiro, las nuevas armas y piezas de artillería eran inútiles para los artilleros. El BRL empleó a más de doscientas personas, la mayoría mujeres, que utilizando calculadoras de mesa resolvían las ecuaciones

balísticas necesarias. La preparación de las tablas para una sola arma le habría llevado a una persona muchas horas, incluso días.

Mauchly, un catedrático de Ingeniería Eléctrica de la Universidad de Pennsylvania, y Eckert, uno de sus alumnos de licenciatura, propusieron construir un computador de propósito general usando tubos de vacío para utilizarlo en las aplicaciones de la BRL. En 1943 esta proposición fue aceptada por el ejército y se comenzó a trabajar en el ENIAC. La máquina que construyeron era enorme, pesaba treinta toneladas, ocupaba 15 000 pies cuadrados y contenía más de 18 000 tubos de vacío. Cuando funcionaba consumía 140 Kilowatios de potencia. También era bastante más rápida que cualquier computador electromecánico, ya que era capaz de efectuar 5 000 sumas por segundo.

El ENIAC era una máquina decimal y no binaria. Es decir, los números estaban representados en forma decimal y la aritmética se realizaba también en el sistema decimal. Su memoria consistía en veinte «acumuladores», cada uno capaz de contener un número decimal de diez dígitos. Cada dígito estaba representado por un anillo de diez tubos de vacío. En un momento dado, solo uno de los tubos de vacío estaba en estado ON, representando uno de los diez dígitos. Uno de los mayores inconvenientes del ENIAC era que tenía que ser programado manualmente mediante conmutadores y conectando y desconectando cables.

El ENIAC se terminó en 1946, demasiado tarde para ser utilizado durante la guerra. En su lugar, su primera misión fue realizar una serie de cálculos complejos que se usaron para ayudar a determinar la viabilidad de la bomba de hidrógeno. El uso del ENIAC para una función distinta de aquella para la que fue construido demostró su naturaleza de propósito general. Así, 1946 marcó el comienzo de la nueva era de los computadores electrónicos, culminando años de esfuerzo. El ENIAC siguió funcionando bajo la dirección del BRL hasta 1955, cuando fue desmontado.

La máquina de von Neumann La tarea de cargar y modificar programas para el ENIAC era extremadamente tediosa. El proceso de programación podría ser más fácil si el programa se representara en una forma adecuada para ser guardado en la memoria junto con los datos. Entonces, un computador podría conseguir sus instrucciones leyéndolas de la memoria, y se podría hacer o modificar un programa colocando los valores en una zona de memoria.

Esta idea conocida como *concepto del programa-almacenado*, se atribuye a los diseñadores del ENIAC, sobre todo al matemático John von Neumann, que era asesor del proyecto ENIAC. La idea fue también desarrollada aproximadamente al mismo tiempo por Turing. La primera publicación de la idea fue en una propuesta de von Neumann para un nuevo computador en 1945, el EDVAC (*Electronic Discrete Variable Computer*).

En 1946 von Neumann y sus colegas empezaron, en el Instituto para Estudios Avanzados de Princeton, el diseño de un nuevo computador de programa-almacenado, que llamaron IAS. El computador IAS, no completado hasta 1952, es el prototipo de toda una serie de computadores de propósito general.

La Figura 2.1 muestra la estructura general del computador IAS. Esta consta de:

- Una memoria principal que almacena tanto datos como instrucciones [1].

[1] En este libro, a menos que se diga lo contrario, el término instrucción hace referencia a una instrucción máquina que es directamente interpretada y ejecutada por el procesador, a diferencia de una instrucción de un lenguaje de alto nivel tal como Ada o Pascal, que previamente a ser ejecutada tiene que ser compilada en una serie de instrucciones máquina.

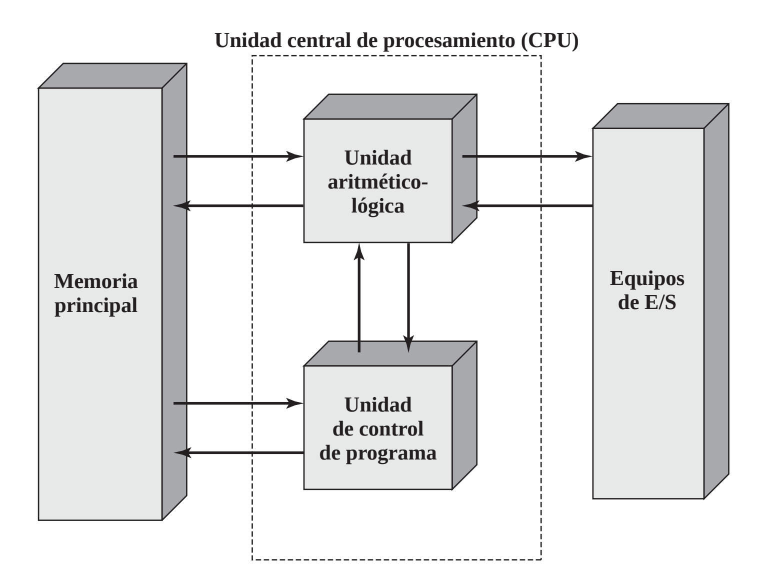


Figura 2.1. Estructura del computador IAS.

- Una unidad aritmético-lógica (ALU) capaz de hacer operaciones con datos binarios.
- Una unidad de control que interpreta las instrucciones en memoria y provoca su ejecución.
- Un equipo de entrada salida (E/S) dirigido por la unidad de control.

Esta estructura fue esbozada en la primera proposición de von Neumann, que merece la pena mencionar en este momento [VONM45]:

> 2.2. **Primero:** como el dispositivo es principalmente un computador, tendrá que realizar las operaciones aritméticas elementales muy frecuentemente. Estas son la suma, la resta, la multiplicación y la división: $+, -, \times, \div$. Es por tanto razonable que contenga elementos especializados solo en estas operaciones.
>
> Debe observarse, sin embargo, que aunque este principio parece consistente, la manera específica de cómo se aplica requiere un examen cuidadoso... En cualquier caso, tendrá que existir la parte de *aritmética central* que constituirá la *primera parte específica:* CA (*Central Arithmetical*).
>
> 2.3. **Segundo**: el control lógico del dispositivo, es decir, la secuenciación adecuada de las operaciones, debe ser realizado eficientemente por un órgano de control central. Si el dispositivo tiene que ser *versátil*, es decir, lo más cercano posible a servir para *todo uso*, entonces hay que hacer una distinción entre las instrucciones específicas que se dan y definir un problema particular, y los órganos de control general que se ocupan de que se lleven a cabo estas instrucciones —sean cuales sean—. Las primeras deben almacenarse en algún lugar; las otras deben representarse definiendo partes operativas del dispositivo. Con el *control central* nos referimos solo a esta última función, y los órganos que la realizan forman la *segunda parte específica:* CC (*Central Control*).
>
> 2.4. **Tercero**: cualquier dispositivo que realice secuencias largas y complicadas de operaciones (concretamente de cálculo) debe tener una memoria considerable [...].
>
> (b) Las instrucciones que gobiernan un problema complicado pueden constituir un material considerable, sobre todo si el código es circunstancial (lo cual ocurre en la mayoría de las situaciones).

Este material debe tenerse en cuenta [...].

En cualquier caso, la *memoria* total es *la tercera parte específica del dispositivo: M* (Memoria).

2.6. Las tres partes específicas CA, CC (juntas C) y M corresponden a las neuronas *asociativas* del sistema nervioso humano. Queda por discutir los equivalentes a las neuronas *sensiorales* o *aferentes* y las *motoras* o *eferentes*. Estos son los órganos del dispositivo de *entrada y salida* [...].

El dispositivo tiene que estar dotado con la habilidad de mantener contacto de entrada y salida (sensorial y motor) con medios específicos de este tipo (cf.1.2): el medio será llamado el *medio de grabación exterior del dispositivo: R* (*Recording*) [...].

2.7. **Cuarto**: el dispositivo tiene que tener órganos para transferir [...] información a partir de R a sus partes específicas C y M. Estos órganos forman su *entrada,* la *cuarta parte específica: I* (*Input*). Veremos que lo mejor es hacer todas las transferencias a partir de R (mediante I) hasta M y nunca directamente a partir de C [...].

2.8. **Quinto**: El dispositivo tiene que tener órganos para transferir [...] información a partir de sus partes específicas C y M hacia R. Estos órganos forman su *salida, la quinta parte específica: O* (*Output*). Veremos que es mejor, de nuevo, hacer todas las transferencias a partir de M (mediante O) a R, y nunca directamente a partir de C [...].

Salvo raras excepciones, todos los computadores de hoy en día tienen la misma estructura general y funcionamiento que la indicada en las máquinas de von Neumann. Por tanto, merece la pena en este momento describir brevemente la manera de operar del computador IAS [BURK46]. Siguiendo [HAYE88], la terminología y la notación de von Neumann han cambiado para ajustarse más a las necesidades actuales; los ejemplos e ilustraciones que acompañan a esta exposición están basados en el último texto.

La memoria del IAS consiste en 1 000 posiciones de almacenamiento, llamadas *palabras*, de cuarenta dígitos binarios (bits) cada una [2]. Tanto los datos como las instrucciones se almacenan ahí. Por tanto, los números se pueden representar en forma binaria y cada instrucción tiene también un código binario. La Figura 2.2 muestra estos formatos. Cada número se representa con un bit de signo y 39 bits de valor. Una palabra puede contener también dos instrucciones de veinte bits, donde cada instrucción consiste en un código de operación de ocho bits (codop) que especifica la operación que se va a realizar y una dirección de doce bits que indica una de las palabras de la memoria (numeradas de 0 a 999).

La unidad de control dirige el IAS captando instrucciones de la memoria y ejecutando una a una. Para explicar esto, se necesita un diagrama de estructura más detallado, como se indica en la Figura 2.3. Esta figura muestra que tanto la unidad de control como la ALU contienen posiciones de almacenamiento, llamadas *registros*, definidos de la siguiente manera:

- **Registro Temporal de Memoria (MBR, *Memory Buffer Register*):** contiene una palabra que debe ser almacenada en la memoria, o es usado para recibir una palabra procedente de la memoria.

[2] No hay una definición universal del término *palabra*. En general, una palabra es un conjunto ordenado de bytes o bits que representa la unidad básica de almacenamiento de información que se puede almacenar, transmitir o con la que se puede operar en un determinado computador. Normalmente, si un procesador tiene un conjunto de instrucciones de longitud fija, entonces la longitud de las instrucciones es igual a la longitud de palabra.

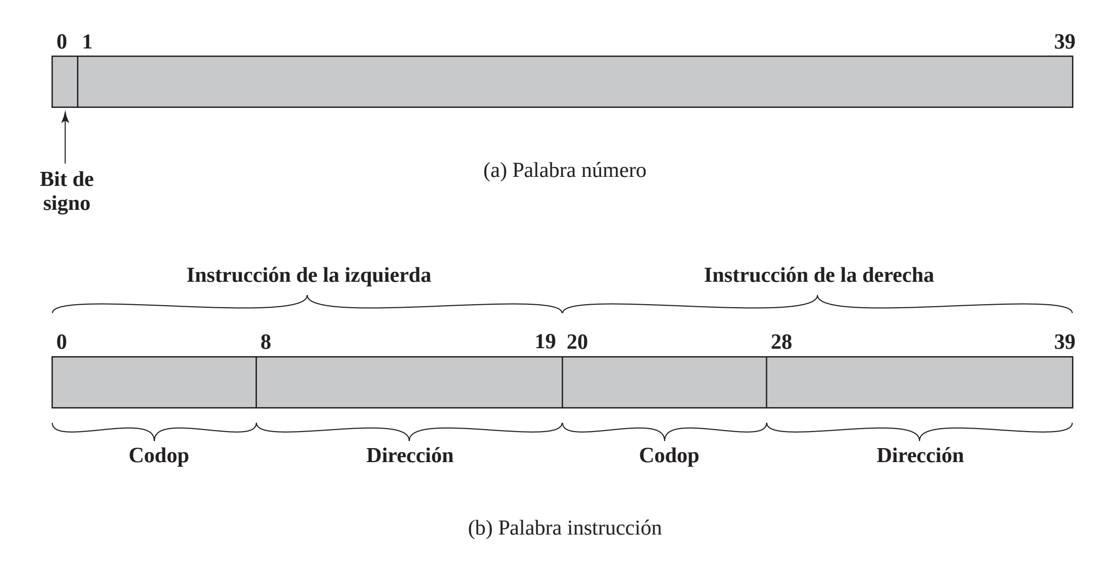


Figura 2.2. Formatos de memoria IAS.

- **Registro de Dirección de Memoria (MAR, *Memory Address Register*):** especifica la dirección en memoria de la palabra que va a ser escrita o leída en MBR.
- **Registro de Instrucción (IR, *Instruction Register*):** contiene los ocho bits del código de operación de la instrucción que se va a ejecutar.
- **Registro Temporal de Instrucción (IBR, *Instruction Buffer Register*):** empleado para almacenar temporalmente la instrucción contenida en la parte derecha de una palabra en memoria.
- **Contador de Programa (PC, *Program Counter*):** contiene la dirección de la próxima pareja de instrucciones que van a ser captadas de la memoria.
- **Acumulador (AC) y Multiplicador cociente (MQ, *Multiplier Quotient*):** Se emplean para almacenar operandos y resultados de operaciones de la ALU temporalmente. Por ejemplo, el resultado de multiplicar dos números de cuarenta bits es un número de ochenta bits; los cuarenta bits más significativos se almacenan en el AC y los menos significativos en el MQ.

El IAS opera ejecutando repetidamente un *ciclo instrucción*, como se puede ver en la Figura 2.4. Cada ciclo instrucción consta de dos subciclos. Durante el *ciclo de captación*, el codop de la siguiente instrucción es cargado en el IR y la parte que contiene la dirección es almacenada en el MAR. Esta instrucción puede ser captada desde el IBR, o puede ser obtenida de la memoria cargando una palabra en el MBR, y luego en IBR, IR y MAR.

¿Por qué la indirección? Todas estas operaciones están controladas por circuitos electrónicos, y dan lugar al uso de caminos de datos. Para simplificar la electrónica, se usa un solo registro para especificar la dirección en memoria para lectura o escritura, y un solo registro para la fuente o el destino.

Una vez que el codop está en el IR, se lleva a cabo el *ciclo de ejecución*. Los circuitos de control interpretan el codop y ejecutan la instrucción enviando las señales de control adecuadas para provocar que los datos se transfieran o que la ALU realice una operación.

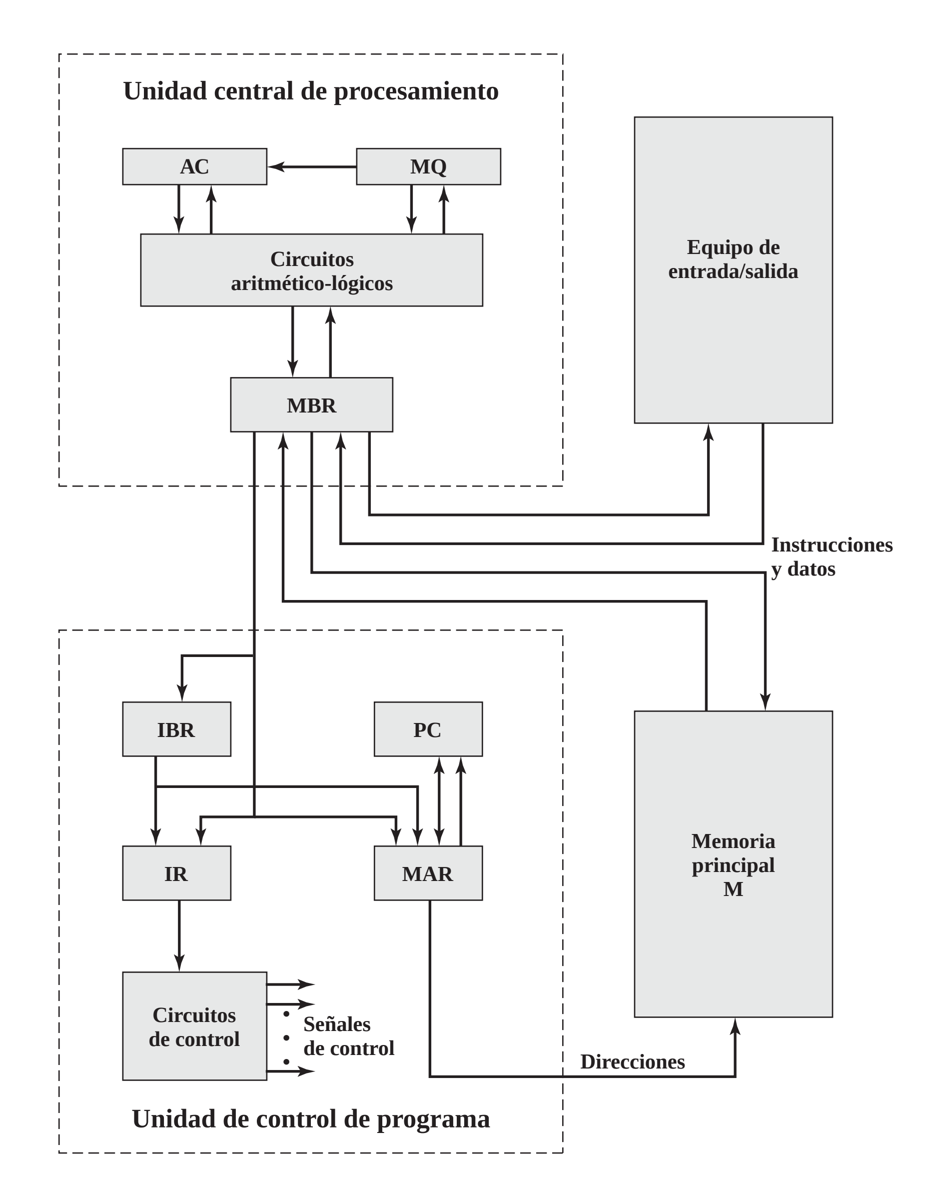


Figura 2.3. Estructura ampliada del computador IAS.

El computador IAS tiene un total de 21 instrucciones, que se indican en la Tabla 2.1. Estas se pueden agrupar de la siguiente manera:

- **Transferencia de datos:** transferir datos entre la memoria y los registros de la ALU o entre dos registros de la ALU.
- **Salto incondicional:** normalmente la unidad de control ejecuta instrucciones secuencialmente en la memoria. Las instrucciones de salto pueden cambiar esta secuencialidad. Esto facilita las operaciones repetitivas.
- **Salto condicional:** el salto depende de una condición, lo que permite puntos de decisión.

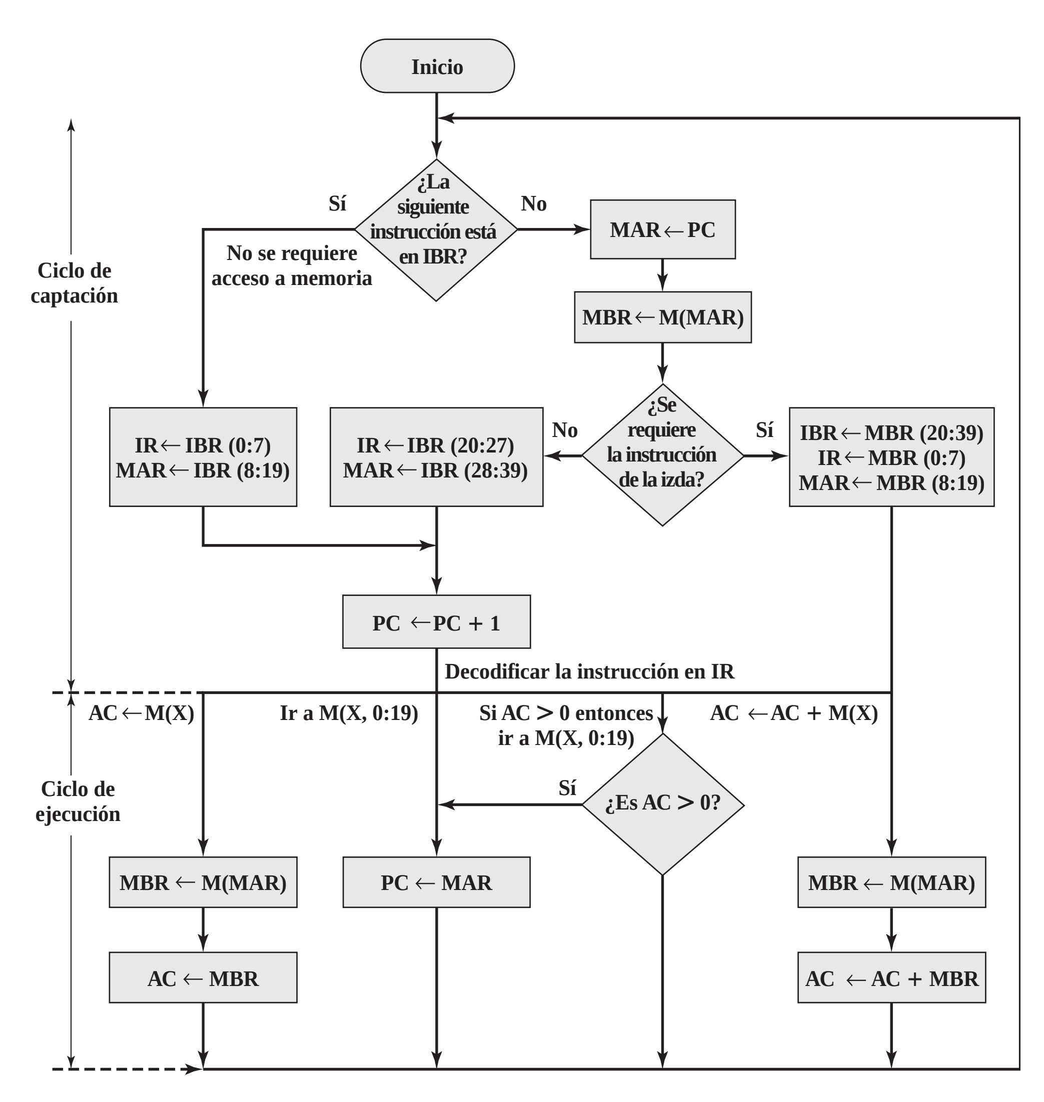


Figura 2.4. Diagrama de flujo parcial de las operaciones del IAS.

- **Aritmética:** operaciones realizadas por la ALU.
- **Modificación de direcciones:** permite que la ALU haga operaciones con las direcciones y las inserte en instrucciones almacenadas en memoria. Esto permite una considerable flexibilidad de direccionamiento en un programa.

La Tabla 2.1 presenta las instrucciones en una forma simbólica y fácil de leer. En realidad, cada instrucción debe tener el formato de la Figura 2.2b. La parte de codop (los ocho primeros bits) especifican cual de las 21 instrucciones va a ser ejecutada. La parte de la dirección (los doce bits restantes) especifican cual de las 1 000 posiciones de memoria está implicada en la ejecución de la instrucción.

Tabla 2.1. El conjunto de instrucciones del IAS.

Tipo de instrucción	Codop	Representación simbólica	Descripción
Transferencia de datos	00001010	LOAD MQ	Transferir el contenido del registro MQ al acumulador AC
	00001001	LOAD MQ, M(X)	Transferir el contenido de la posición de memoria X a MQ
	00100001	STOR M(X)	Transferir el contenido del acumulador a la posición de memoria X
	00000001	LOAD M(X)	Trasferir M(X) al acumulador
	00000010	LOAD – M(X)	Transferir – M(X) al acumulador
	00000011	LOAD |M(X)|	Transferir el valor absoluto de M(X) al acumulador
	00000100	LOAD – |M(X)|	Transferir – |M(X)| al acumulador
Salto incondicional	00001101	JUMP M(X,0:19)	Captar la siguiente instrucción de la mitad izquierda de M(X)
	00001110	JUMP M(X,20:39)	Captar la siguiente instrucción de la mitad derecha de M(X)
Salto condicional	00001111	JUMP + M(X,0:19)	Si el número en el acumulador es no negativo, captar la siguiente instrucción de la mitad izquierda de M(X)
	00010000	JUMP + M(X,20:39)	Si el número en el acumulador es no negativo, captar la siguiente instrucción de la mitad derecha de M(X)
Aritmética	00000101	ADD M(X)	Sumar M(X) a AC; colocar el resultado en AC
	00000111	ADD |M(X)|	Sumar |M(X)| a AC; colocar el resultado en AC
	00000110	SUB M(X)	Restar M(X) a AC; colocar el resultado en AC
	00001000	SUB |M(X)|	Restar |M(X)| a AC; colocar el resultado en AC
	00001011	MUL M(X)	Multiplicar M(X) por MQ: colocar los bits más significativos del resultado de AC, y los menos significativos en MQ
	00001100	DIV M(X)	Dividir AC por M(X); colocar el cociente en MQ y el resto en AC
	00010100	LSH	Multiplicar el acumulador por 2; esto es, desplazar su contenido una posición a la derecha
	00010101	RSH	Dividir el acumulador por 2; esto es, desplazar su contenido una posición a la derecha
Modificación de direcciones	00010010	STOR M(X,8:19)	Reemplazar el campo de dirección de la izquierda de M(X) por los 12 bits de la derecha de AC
	00010011	STOR M(X,28:39)	Reemplazar el campo de dirección de la derecha de M(X) por los doce bits de la derecha de AC

La Figura 2.4 muestra varios ejemplos de la ejecución de una instrucción por la unidad de control. Hay que destacar que cada operación requiere varios pasos. Algunas son bastante complejas. ¡La operación de multiplicación requiere 39 suboperaciones, una para cada bit excepto para el bit de signo!

Computadores comerciales Los años cincuenta contemplaron el nacimiento de la industria de computadores con dos compañías, Sperry e IBM, dominando el mercado.

En 1947 Eckert y Mauchly formaron la Eckert-Mauchly Computer Corporation para fabricar computadores con fines comerciales. Su primera máquina de éxito fue el UNIVAC I (*Universal Automatic Computer*), que fue empleada por la oficina del censo para sus cálculos en 1950. La Eckert-Mauchly Computer Corporation formó luego parte de la división UNIVAC de la Sperry-Rand Corporation, que siguió construyendo una serie de máquinas sucesoras de la primera.

El UNIVAC I fue el primer computador comercial de éxito. Estaba diseñado, como su nombre indica, tanto para aplicaciones científicas como comerciales. El primer documento que describía el sistema mencionaba como ejemplos de tareas que podía realizar operaciones algebraicas con matrices, problemas de estadística, reparto de primas para las compañías de seguros de vida y problemas logísticos.

El UNIVAC II, que tenía una capacidad de memoria mayor y más aplicaciones que el UNIVAC I, salió al mercado al final de los cincuenta e ilustra varias tendencias que han permanecido como características de la industria de computadores. Primera, los avances en la tecnología permiten a las compañías seguir construyendo computadores más grandes y más potentes. Segunda, cada compañía intenta hacer sus nuevas máquinas superiores y compatibles con las anteriores. Esto significa que los programas escritos para las viejas máquinas pueden ejecutarse en las nuevas máquinas. Esta estrategia se adopta para retener la base de clientes; es decir, que cuando un cliente decide comprar una máquina nueva, probablemente la comprará a la misma compañía para evitar perder su inversión en programas.

La división UNIVAC comenzó también el desarrollo de la serie de computadores 1100, que fue su producto principal. Esta serie ilustra una distinción que existió en aquella época. El primer modelo, el UNIVAC 1103, y sus sucesores durante muchos años, estaban diseñados principalmente para aplicaciones científicas que implicaban cálculos largos y complejos. Otras compañías se centraron en el campo de la gestión, lo que conllevaba el procesamiento de grandes cantidades de textos. Esta separación desapareció hace tiempo, pero fue patente durante algunos años.

IBM, que había ayudado a construir el Mark I y era entonces el principal fabricante de equipos de procesamiento con tarjetas perforadas, sacó su primer computador con programas almacenados electrónicamente, el 701, en 1953. El 701 fue diseñado principalmente para aplicaciones científicas [BASH81]. En 1955 IBM presentó los productos 702, que tenían varias características hardware que lo hacían adecuado para aplicaciones de gestión. Estos fueron los primeros de una larga serie de computadores 700/7000 que situaron a IBM como el fabricante de computadores dominante, con gran diferencia.

LA SEGUNDA GENERACIÓN: LOS TRANSISTORES

El primer cambio importante en los computadores electrónicos vino con la sustitución de los tubos de vacío por transistores. El transistor es más pequeño, más barato, disipa menos calor y puede ser usado

Tabla 2.2. Generación de computadores.

Generación	Fechas aproximadas	Tecnología	Velocidad típica (operaciones por segundo)
1	1946-1957	Válvulas	40 000
2	1958-1964	Transistores	200 000
3	1965-1971	Pequeña y media integración	1 000 000
4	1972-1977	Gran integración (LSI)	10 000 000
5	1978-1991	Alta integración (VLSI)	100 000 000
6	1991-	Ultra alta integración (ULSI)	1 000 000 000

de la misma forma que un tubo de vacío en la construcción de computadores. Mientras que un tubo de vacío requiere cables, placas de metal, una cápsula de cristal y vacío, el transistor es un *dispositivo de estado sólido*, hecho con silicio.

El transistor fue inventado en los Laboratorios Bell en 1947 y en los años cincuenta y provocó una revolución electrónica. Sin embargo, los computadores completamente transistorizados no estuvieron disponibles comercialmente hasta el final de los cincuenta. IBM no fue la primera compañía que lanzó esta nueva tecnología. NCR y, con más éxito, RCA fueron los primeros en sacar pequeñas máquinas de transistores. IBM los siguió pronto con la serie 7000.

El uso del transistor define la *segunda generación* de computadores. La clasificación de los computadores en generaciones basándose en la tecnología hardware empleada fue ampliamente aceptada (Tabla 2.2). Cada nueva generación se caracteriza por la mayor velocidad, mayor capacidad de memoria y menor tamaño que la generación anterior.

También hay otros cambios. En la segunda generación se introdujeron unidades lógicas y aritméticas y unidades de control más complejas, el uso de lenguajes de programación de alto nivel, y se proporcionó un *software del sistema* con el computador.

La segunda generación es destacable también por la aparición de la empresa Digital Equipment Corporation (DEC). DEC fue fundada en 1957 y en este año sacó su primer computador, el PDP-1. Este computador y esta compañía iniciaron el desarrollo de los minicomputadores que fue de gran importancia en la tercera generación.

El IBM 7094 Desde la introducción en 1952 de la serie 700 y la introducción del último miembro en 1964 de la serie 7000, esta línea de productos sufrió la evolución típica de los computadores. Los productos sucesivos de la línea presentaron un aumento de prestaciones y capacidad y/o la disminución de precios.

Esta tendencia se puede ver en la Tabla 2.3. El tamaño de la memoria principal, en múltiplos de 2^{10} palabras de 36 bits, creció de 2K (1K = 2^{10}) a 32K palabras [3], mientras que el tiempo de acceso a una palabra de memoria, el *tiempo de ciclo de memoria*, cayó de 30 μs a 1,4 μs. El número de códigos de operación creció de un modesto 24 a 185.

---

[3] El uso de prefijos numéricos, como kilo y giga, se explica y trata en un documento de la web de Recursos del Estudiante de Informática en WilliamStallings.com/StudentSupport.html.

Tabla 2.3. Ejemplos de miembros de la serie IBM 700/7000.

Modelos	Primera entrega	Tecnología de la CPU	Tecnología de la memoria	Tiempo de ciclo (μs)	Tamaño de memoria (K)	Número de codops	Número de registros índice	Punto flotante por hardware	Solapamiento de E/S (Canales)	Solapamiento de captación instrucciones	Velocidad de captación (relativa a 701)
701	1952	Tubos de vacío	Tubos electro-está.	30	2-4	24	0	no	no	no	1
704	1955	Tubos de vacío	Núcleos	12	4-32	80	3	sí	no	no	2,5
709	1958	Tubos de vacío	Núcleos	12	32	140	3	sí	sí	no	4
7090	1960	Transistor	Núcleos	2,18	32	169	3	sí	sí	no	25
7094.I	1962	Transistor	Núcleos	2	32	185	7	sí (doble precisión	sí	sí	30
7094.II	1964	Transistor	Núcleos	1,4	32	185	7	sí (doble precisión	sí	sí	50

La columna final indica la velocidad de ejecución relativa de la CPU. El incremento de velocidad se logró mejorando la electrónica (por ejemplo, una implementación con transistores es más rápida que con tubos de vacío) y con una circuitería más compleja. Por ejemplo, el IBM 7094 incluye un registro de respaldo de instrucciones, usado como buffer de la siguiente instrucción. La unidad de control capta las dos palabras adyacentes de la memoria para captar una instrucción. Excepto en una instrucción de salto, que suele ser poco frecuente, esto significa que la unidad de control tiene que acceder a la memoria en busca de una instrucción en solo la mitad de los ciclos de instrucción. Esta precaptación reduce considerablemente el tiempo medio de ciclo de instrucción.

El significado del resto de las columnas de la Tabla 2.3 es claro tras la explicación anterior.

La Figura 2.5 muestra una configuración (con muchos periféricos) del IBM 7094, que es representativo de los computadores de la segunda generación [BELL71a]. Merece la pena señalar varias diferencias con el computador IAS. La más importante es el uso de *canales de datos*. Un canal de datos es un módulo de E/S independiente con su propio procesador y su propio conjunto de instrucciones. En un computador con tales dispositivos, la CPU no ejecuta instrucciones detalladas de E/S. Tales instrucciones son almacenadas en una memoria principal para ser ejecutadas con un procesador de uso específico para el canal de datos mismo. La CPU inicia una transferencia de E/S enviando señales de control al canal de datos, instruyéndolo para ejecutar una secuencia de instrucciones en

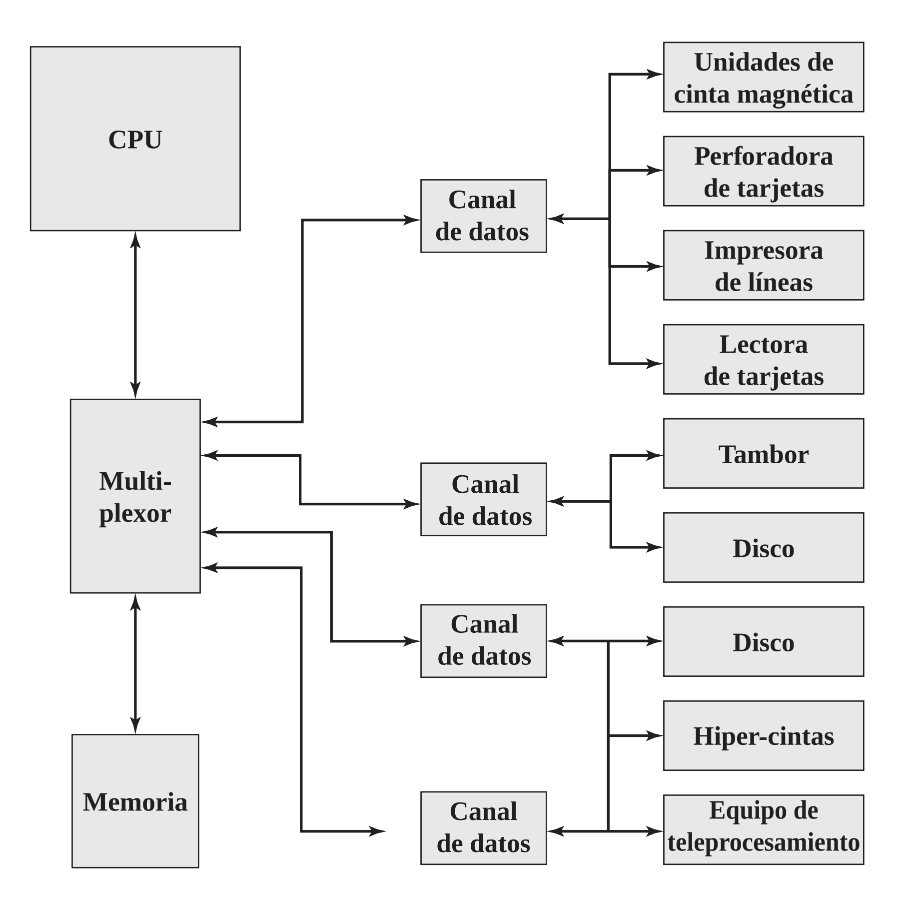


Figura 2.5. Configuración de un IBM 7094.

memoria. El canal de datos realiza esta tarea independientemente de la CPU y de las señales de la CPU hasta que la operación se completa. Esta disposición libera a la CPU de una carga de procesamiento considerable.

Otra característica es el *multiplexor*, que es el punto de conexión central de los canales de datos, la CPU y la memoria. El multiplexor organiza los accesos a la memoria desde la CPU y los canales de datos, permitiendo a estos dispositivos actuar de forma independiente.

LA TERCERA GENERACIÓN: LOS CIRCUITOS INTEGRADOS

A un transistor simple y autocontenido se le llama *componente discreto*. A través de los años cincuenta y principios de los sesenta, los equipos electrónicos estaban compuestos en su mayoría por componentes discretos —transistores, resistencias, capacidades, etc.—. Los componentes discretos se fabricaban separadamente, encapsulados en sus propios contenedores, y soldados o cableados juntos en tarjetas de circuitos en forma de panel, que eran instalados en computadores, osciloscopios y otros equipos electrónicos. Cuando un dispositivo necesitaba un transistor, había que soldar este, que tenía una forma de un pequeño tubo de metal y contenía una pieza de silicio del tamaño de la cabeza de un alfiler, en una tarjeta de circuitos. Todo el proceso de fabricación desde el transistor hasta el panel de circuitos era caro y engorroso.

Estos hechos fueron el comienzo del surgimiento de problemas en la industria de computadores. Los primeros computadores de la segunda generación contenían alrededor de 10 000 transistores. Esta cantidad creció a cientos de miles, haciendo cada vez más difícil la fabricación de máquinas nuevas y más potentes.

En 1958 ocurrió algo que revolucionó la electrónica y comenzó la era de la microelectrónica: la invención del circuito integrado. El circuito integrado define la tercera generación de computadores. En esta sección haremos una breve introducción a la tecnología de circuitos integrados. Después veremos los que quizá sean los dos miembros más importantes de la tercera generación, que surgieron al principio de la era: el IBM Sistema/360 y el DEC PDP-8.

Microelectrónica Microelectrónica significa literalmente «pequeña electrónica». Desde los comienzos de la electrónica digital y la industria de computadores, ha habido una tendencia persistente y consistente hacia la reducción del tamaño de los circuitos electrónicos digitales. Antes de examinar las implicaciones y beneficios de esta tendencia, necesitamos decir algo sobre la naturaleza de la electrónica digital. En el Apéndice A se encuentra una discusión más detallada.

Los elementos básicos de un computador digital, como ya sabemos, deben ofrecer almacenamiento, procesamiento y control de funciones. Solo se requieren dos tipos fundamentales de componentes (Figura 2.6): puertas y celdas de memoria. Una puerta es un dispositivo que implementa una función lógica o booleana simple, como SI *A* AND *B* ES CIERTO ENTONCES *C* ES CIERTO (puerta AND). A tales dispositivos se les llama puertas porque controlan el flujo en cierta manera, como lo hacen las puertas de un canal. La celda de memoria es un dispositivo que puede almacenar un dato de un bit; es decir, el dispositivo puede estar, en un instante dado, en uno de dos estados estables. Interconectando muchos de estos dispositivos fundamentales, podemos construir un computador. Podemos relacionar esto con nuestras cuatro funciones básicas de la siguiente forma:

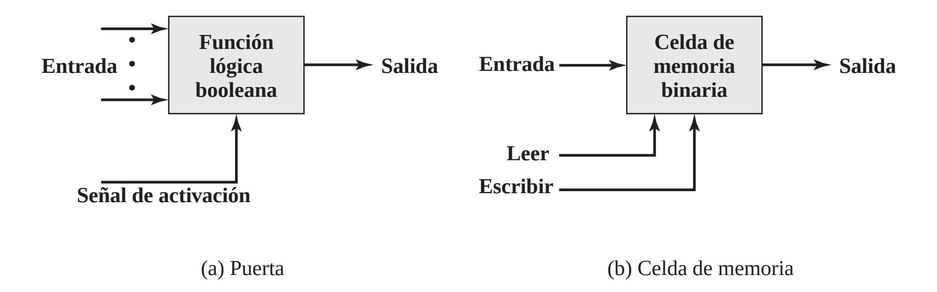


Figura 2.6. Elementos de un computador básico.

- **Almacén de datos:** proporcionado por las celdas de memoria.
- **Procesamiento de datos:** proporcionado por las puertas.
- **Transferencias de datos:** los caminos entre componentes se usan para llevar datos de memoria a memoria y de memoria, a través de las puertas, a memoria.
- **Control:** los caminos entre componentes pueden llevar las señales de control. Por ejemplo, una puerta tendrá dos entradas de datos más una entrada de control que activará la puerta. Cuando la señal de control está en ON, la puerta realiza su función con los datos de entrada y produce un dato de salida. De manera similar, las celdas de memoria almacenarán el bit en su entrada si la señal de control WRITE está ON y situarán el bit en la salida cuando la señal de control READ esté ON.

Por tanto, un computador consta de puertas, celdas de memoria e interconexiones entre estos elementos. Las puertas y las celdas de memoria están constituidas por componentes electrónicos simples.

Los circuitos integrados utilizaron el hecho de que componentes como transistores, resistencias y conductores podían ser fabricados a partir de un semiconductor como el silicio. Es simplemente un avance del arte del estado sólido consistente en fabricar un circuito entero en un pequeño trozo de silicio, en vez de ensamblar componentes discretos hechos a partir de trozos de silicio separados en el mismo circuito. Se pueden construir cientos e incluso miles de transistores al mismo tiempo en una sola oblea de silicio. Igualmente importante es que estos transistores pueden ser conectados con un proceso de metalización para formar circuitos.

La Figura 2.7 muestra los conceptos clave de un circuito integrado. Se divide una fina *oblea* de silicio en una matriz de pequeñas áreas, cada una de unos pocos milímetros cuadrados. Se fabrica el mismo patrón de circuito en cada área, y la oblea se divide en *chips*. Cada chip consiste en muchas puertas más una serie de puntos para conexiones de entrada y salida. El chip es encapsulado en una carcasa que lo proteje y proporciona patas para conectar dispositivos fuera del chip. Varios de estos elementos pueden ser interconectados en una tarjeta de circuito impreso para producir circuitos más complejos y mayores.

Inicialmente solo podían fabricarse y encapsularse juntas, con fiabilidad, unas pocas puertas o celdas de memoria. A estos primeros circuitos integrados se les llama de *pequeña escala de integración* (SSI, *Small-Scale Integration*). A medida que el tiempo pasó, fue posible encapsular más y más componentes en un mismo chip. Este crecimiento en densidad se puede ver en la Figura 2.8; esta es

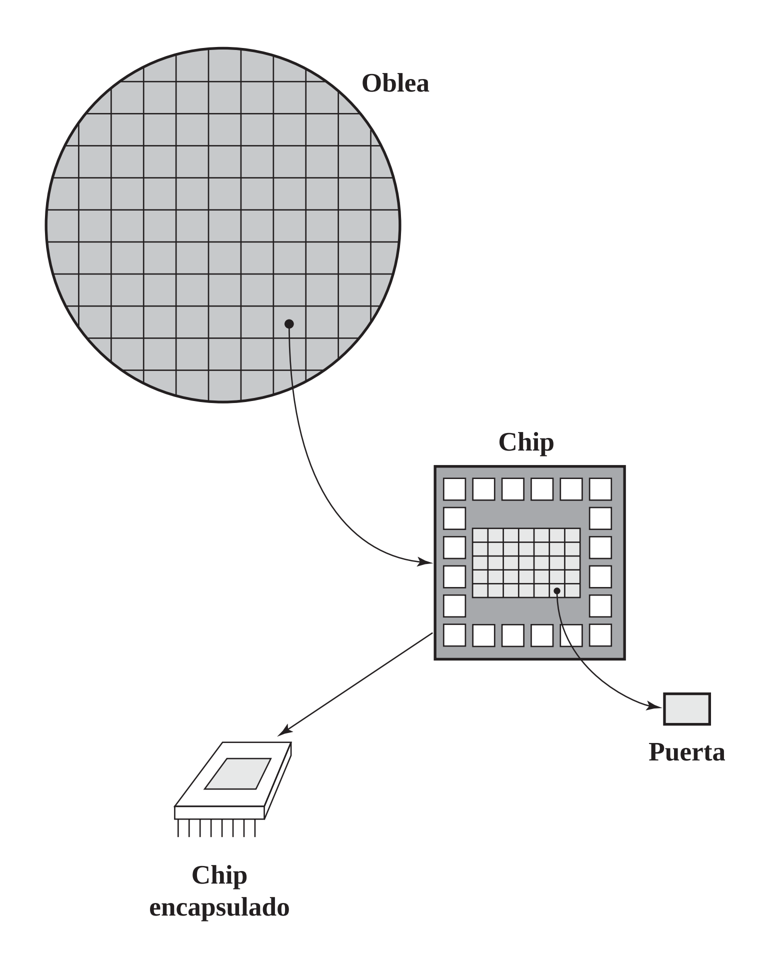


Figura 2.7. Relación entre oblea, chip y puerta.

una de las tendencias tecnológicas más importantes que nunca se han visto [4]. Esta figura refleja la famosa ley de Moore, que fue propuesta por Gordon Moore, cofundador de Intel, en 1965 [MOOR65]. Moore observó que el número de transistores que se podrían integrar en un solo chip se duplicaba cada año y se predecía correctamente que esto continuaría en un futuro cercano. Para sorpresa de muchos, incluido Moore, este ritmo continuaría año tras año y década tras década. El ritmo disminuyó duplicándose cada dieciocho meses en los setenta, pero ha mantenido esta velocidad desde entonces.

Las consecuencias de la ley de Moore son profundas:

1. El precio de un chip ha permanecido prácticamente invariable a través de este periodo de rápido crecimiento en densidad. Esto significa que el coste de la lógica del computador y de la circuitería de la memoria ha caído a una velocidad drástica.

2. Ya que los elementos de la lógica y la memoria están más próximos en chips más densamente encapsulados, la longitud de las interconexiones eléctricas ha disminuido, incrementándose así la velocidad operativa.

---

[4] Nótese que el eje vertical utiliza una escala logarítmica. En el documento de repaso de matemáticas en la web de Recursos del Estudiante de Informática en WilliamStallings.com/StudentSupport.html se hace un repaso básico de las escalas logarítmicas.

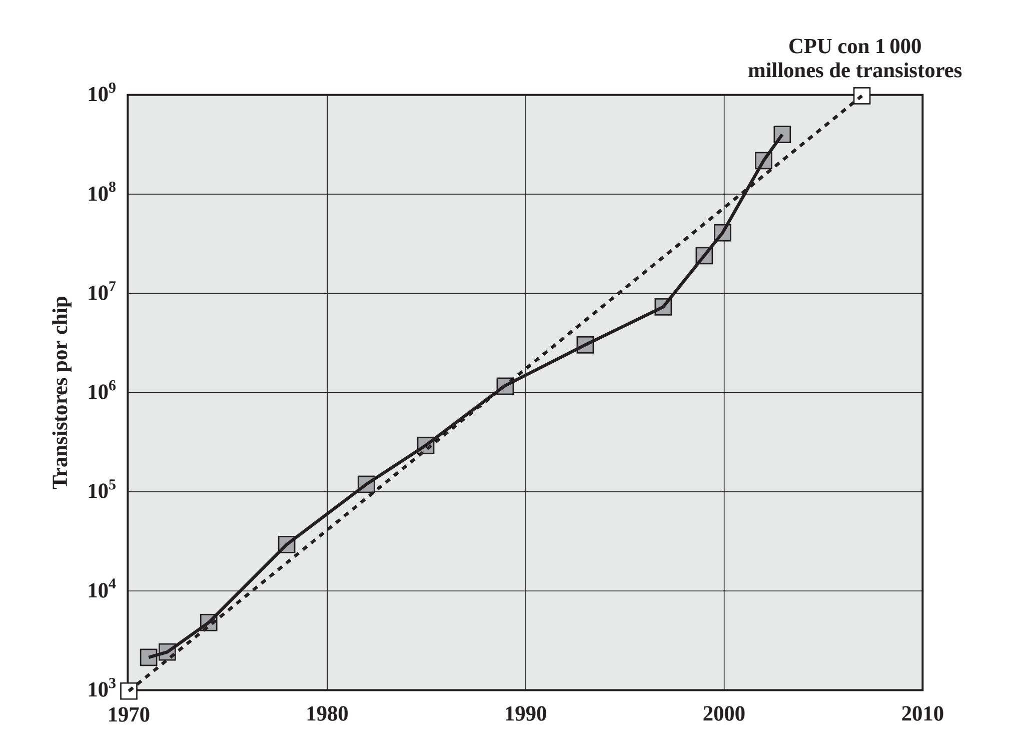


Figura 2.8. Crecimiento en el número de transistores en la CPU [BOHR03].

3. El computador es ahora más pequeño, lo que lo hace más adecuado para más entornos.
4. Hay una reducción de las necesidades de potencia y refrigeración.
5. Las interconexiones de los circuitos integrados son mucho más fiables que las conexiones soldadas. Con más circuitos en cada chip hay menos conexiones entre chips.

En 1964, IBM tenía un firme dominio del mercado con sus máquinas de la serie 7000. Aquel año, IBM anunció el Sistema/360, una nueva familia de productos de computadores. Aunque el anuncio mismo no fue ninguna sorpresa, contenía algunas noticias desagradables para los clientes habituales de IBM: la línea de productos 360 era incompatible con las máquinas IBM anteriores. Por ello la transición al 360 sería difícil para los clientes de IBM. Este fue un paso audaz de IBM, pero sentían que era necesario romper con algunas de las limitaciones de la arquitectura 7000 y producir un sistema capaz de evolucionar junto con la nueva tecnología de circuitos integrados [PADE81, GIFF87]. La estrategia resultó provechosa tanto técnica como financieramente. El 360 fue el éxito de la década y consolidó a IBM como el dominante absoluto en las ventas de computadores, con una cuota de mercado por encima del setenta por ciento. Y, con algunas modificaciones y ampliaciones, la arquitectura del 360 permanece hasta hoy en día en la arquitectura de los grandes computadores (*mainframe*) de IBM [5]. A lo largo del texto se pueden encontrar ejemplos que utilizan esta arquitectura.

---

[5] El término *mainframe* (gran computador), se usa para designar a los computadores más grandes y potentes, después de los supercomputadores. Las características típicas de un gran computador son que soporta grandes bases de datos, tienen un hardware de E/S elaborado, y se usan para procesamiento de datos centralizados.

El Sistema/360 fue la primera *familia* de computadores de la historia que se planeó. La familia abarcaba un amplio rango de prestaciones y precios. La Tabla 2.4 indica alguna de las características clave de los distintos modelos en 1965 (cada miembro de la familia se distingue por un número de modelo). Los distintos modelos eran compatibles en el sentido de que un programa escrito para un modelo, tenía que ser capaz de ser ejecutado por otro modelo de la serie, con la única diferencia del tiempo de ejecución.

El concepto de familia de computadores compatibles era a la vez novedoso y extremadamente exitoso. Un cliente con necesidades modestas y un presupuesto limitado podía empezar con el modelo 30, relativamente barato. Más tarde, si las necesidades del cliente crecían, era posible pasarse a una máquina más rápida, con más memoria, sin sacrificar la inversión ya realizada en software. Las características de una familia son:

- **Conjunto de instrucciones similar o idéntico:** en muchos casos, se encuentran exactamente el mismo conjunto de instrucciones máquina en todos los miembros de la familia. Así, un programa que se ejecuta en una máquina, se podrá ejecutar en cualquier otra. En algunos casos el computador más bajo de la familia tiene un conjunto de instrucciones que es un subconjunto del computador más alto de la familia. Esto quiere decir que los programas se pueden mover hacia arriba pero no hacia abajo.
- **Sistemas operativos similares o idénticos:** el mismo sistema operativo básico está disponible para todos los miembros de la familia. En algunos casos, se añaden características complementarias a los miembros más altos.
- **Velocidad creciente:** la velocidad de ejecución de las instrucciones se incrementa conforme se sube desde los miembros más bajos a los más altos de la familia.
- **Número creciente de puertos de E/S:** conforme se va desde lo más bajo a los más alto de la familia.
- **Tamaño de memoria creciente:** conforme se va de lo más bajo a lo más alto de la familia.
- **Coste creciente:** conforme se va de lo más bajo a lo más alto de la familia.

¿Cómo podría implementarse tal concepto de familia? Las diferencias entre los modelos se basaron en tres factores: la velocidad básica, el tamaño y el grado de simultaneidad [STEV64]. Por ejemplo, podría lograrse mayor velocidad en la ejecución de una instrucción dada usando una circuitería

Tabla 2.4. Características clave de la familia Sistema/360.

Características	Modelo 30	Modelo 40	Modelo 50	Modelo 65	Modelo 75
Tamaño máximo de memoria (bytes)	64K	256K	256K	512K	512K
Velocidad de transferencia de datos procedentes de la memoria (MB/segundo)	0,5	0,8	2,0	8,0	16,0
Tiempo de ciclo del procesador (μ/segundo)	1,0	0,625	0,5	0,25	0,2
Velocidad relativa	1	3,5	10	21	50
Número máximo de canales de datos en un canal (KB/segundo)	250	400	800	1.250	1.250

más compleja en la ALU, permitiendo que las suboperaciones se llevaran a cabo en paralelo. Otro modo de incrementar la velocidad era incrementar la amplitud del camino de los datos entre la memoria principal y la CPU. En el Modelo 30, solo se podía captar un byte (8 bits) a la vez de la memoria principal, mientras que en el Modelo 75 se podían captar ocho bytes a la vez.

El Sistema /360 no solamente dictó la carrera hacia el futuro de IBM, sino también tuvo un profundo impacto en toda la industria. Muchas de sus características se han convertido en un estándar para otros grandes computadores.

DEC PDP-8 En el mismo año que IBM lanzó su primer Sistema/360 tuvo lugar otro lanzamiento trascendental: el PDP-8 de DEC. En aquella época, cuando la mayoría de los computadores requerían una habitación con aire acondicionado, el PDP-8 (llamado por la industria minicomputador en honor a la minifalda de aquellos tiempos) era lo bastante pequeño para ser colocado en lo alto de una mesa de laboratorio o embutido en otro equipo. No podía hacer todo lo que hacían los grandes computadores, pero a 16 000 dólares era suficientemente barato para que cada técnico de laboratorio tuviera uno. Por contra, los computadores de la serie Sistema/360, presentados solo unos meses antes costaban cientos de miles de dólares.

El bajo costo y pequeño tamaño del PDP-8 permitía a otros fabricantes comprarse un PDP-8 e integrarlo en un sistema global para revenderlo. Estos otros fabricantes se conocían como fabricantes de equipos originales (OEM), y el mercado de OEM llegó a tener y aún tiene la mayor cuota del mercado de computadores.

El PDP-8 fue un éxito inmediato y logró el enriquecimiento de DEC. Esta máquina y los otros miembros de la familia PDP-8 que la siguieron (*véase* Tabla 2.5) lograron un status de producción antes reservado a los computadores IBM, con alrededor de 50 000 máquinas vendidas en los siguientes doce años. Como se dice en la historia oficial de DEC, el PDP-8 «estableció el concepto de minicomputador, abriendo el camino a una industria de miles de millones de dólares». También estableció a DEC como el vendedor de minicomputadores número uno y cuando el PDP-8 alcanzó el fin de su vida útil, DEC era el segundo fabricante de computadores detrás de IBM.

En contraste con la arquitectura de conmutador central (Figura 2.5) usada por IBM en sus sistemas 700/7000 y 360, los últimos modelos del PDP-8 usaban una estructura que ahora es prácticamente universal para minicomputadores y microcomputadores: la estructura de bus. Esto se muestra en la Figura 2.9. El bus PDP-8, llamado Omnibus, consiste en 96 hilos conductores separados, usados para control, direccionamiento y datos. Como todos los componentes del sistema comparten un conjunto de caminos, su uso debe estar controlado por la CPU. Esta arquitectura es altamente flexible, permitiendo conectar módulos al bus para crear varias configuraciones.

ÚLTIMAS GENERACIONES

Más allá de la tercera generación hay menos acuerdo general en la definición de las generaciones de computadores. En la Tabla 2.2 se sugieren las que serían la cuarta y la quinta generación, basadas en los avances de la tecnología de los circuitos integrados. Con la introducción de la integración a gran escala (LSI, *Large-Scale Integration*), podía haber más de 1 000 componentes en un simple chip de circuito integrado. Con la integración a muy gran escala (VLSI, *Very-Large Scale Integration*), se lograron más de 10 000 componentes por chip, y los chips VLSI actuales pueden contener más de 100 000 componentes.

Tabla 2.5. Evolución del PDP-8 [VOEL88].

Modelo	**Primera venta**	**Coste del procesador +4K palabras de 12 bit de memoria (miles de dólares)**	**Velo. transfer. de memoria (Pal./μseg.)**	**Vol. (pies cúbicos)**	**Innovaciones y mejoras**
PDP-8	4/65	16,2	1,26	8,0	Producción automática con conductores enrollados (*wire-wrapping*)
PDP-8/5	9/66	8,79	0,08	3,2	Implementación de instrucciones serie
PDP-8/1	4/68	11,6	1,34	8,0	Circuitos integrados de media escala
PDP-8/L	11/68	7,0	1,26	2,0	Chasis menor
PDP-8/E	3/71	4,99	1,52	2,2	Omnibus
PDP-8/M	6/72	3,69	1,52	1,8	Chasis de la mitad de tamaño y menos ranuras que el 8/E
PDP-8/1	1/75	2,6	1,34	1,2	Memoria de semiconductor; procesador de punto flotante

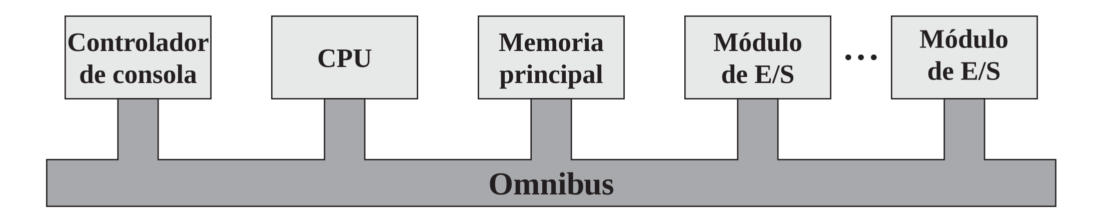


Figura 2.9. Estructura del bus PDP-8.

Con el gran avance de la tecnología, la rápida introducción de nuevos productos, y la importancia del software y las comunicaciones así como del hardware, la clasificación en generaciones se vuelve cada vez menos clara y menos significativa. Se podría decir que la aplicación comercial de nuevos desarrollos resultó uno de los principales cambios de principios de los años setenta y los resultados de estos cambios duran todavía. En esta sección mencionaremos dos de los más importantes.

Memoria semiconductora La primera aplicación de la tecnología de circuitos integrados en computadores dejó a un lado la construcción del procesador (la unidad de control y la unidad aritmético lógica) con chips de circuitos integrados. Sin embargo, se encontró que esta misma tecnología podía usarse para construir memorias.

En los años cincuenta y sesenta, la mayoría de las memorias de los computadores se hacían con pequeños anillos de material ferromagnético, cada uno de un dieciseisavo de pulgada de diámetro. Estos anillos de ferrita se insertaban en mallas de finos cables engarzados en pequeños marcos dentro del computador. Se magnetizaba en un sentido el anillo (llamado *núcleo*) y representaba un uno; magnetizado en el otro sentido, representaba un cero. La memoria de núcleo magnético era más bien rápida; tardaba tan poco como una milésima de segundo en leer un bit almacenado en memoria. Pero era cara, voluminosa y usaba lectura destructiva: el simple hecho de leer un núcleo borraba los datos almacenados en él. Era, por consiguiente, necesario hacer circuitos que recuperaran el dato tan pronto como se extraía.

Entonces, en 1970, Fairchild produjo la primera memoria semiconductora con relativa capacidad. Este chip, del tamaño de un sencillo núcleo de ferrita, podía tener 256 bits de memoria. Era no destructiva y mucho más barata que un núcleo. Tardaba solamente setenta mil millonésimas de segundo en leer un bit. Sin embargo, el coste por bit era mayor que el de un núcleo.

En 1974, ocurrió un hecho sorprendente: el precio por bit de memoria semiconductora cayó por debajo del precio por bit de memoria de núcleo. Siguiendo esto, ha habido una continua disminución del precio de la memoria acompañado de un correspondiente aumento de la densidad de memoria. Esto ha llevado, en pocos años, a hacer máquinas más pequeñas y más rápidas con el mismo tamaño de memoria que máquinas más grandes y más caras. El desarrollo de la tecnología de memorias, junto con el desarrollo de la tecnología de procesadores, del que hablaremos después, cambiaron la naturaleza de los computadores en menos de una década. Aunque los computadores caros y voluminosos permanecieron dentro del panorama, el computador se ha llevado también al «consumidor final», en forma de máquinas de oficina y de computadores personales.

A partir de 1970, la memoria semiconductora ha tenido ocho generaciones: 1K, 4K, 16K, 64K, 256K, 1M, 4M, 16M, 64M, 256M y ahora un giga bits en un solo chip ($1K = 2^{10}$, $1M = 2^{20}$, $1G = 2^{30}$). Cada generación ha proporcionado cuatro veces más densidad de almacenamiento que la generación previa, junto con un menor coste por bit y una mayor velocidad de acceso.

Microprocesadores Igual que la densidad de elementos en los chips de memoria ha continuado creciendo, también lo ha hecho la densidad de elementos de procesamiento. Conforme el tiempo pasaba, en cada chip había más y más elementos, así que cada vez se necesitaban menos y menos chips para construir un procesador de un computador.

En 1971 se hizo una innovación sensacional, cuando Intel desarrolló su 4004. el 4004 fue el primer chip que contenía *todos* los componentes de la CPU en un solo chip: el microprocesador había nacido.

El 4004 podía sumar dos números de cuatro bits y multiplicar solo con sumas sucesivas. Según los estándares de hoy en día, el 4004 es muy primitivo, pero marcó el comienzo de la evolución continua en capacidad y potencia de los microprocesadores.

Esta evolución se puede ver más fácilmente considerando el número de bits que el procesador trata a la vez. No hay una medida clara de esto, pero quizá la mejor medida es la anchura del bus de datos: el número de bits de un dato que pueden venir o ir al procesador a la vez. Otra medida es el número de bits del acumulador o del conjunto de registros de uso general. A menudo, estas medidas coinciden, pero no siempre. Por ejemplo, hay una serie de microprocesadores que operan con números de 16 bits en los registros, pero que solo pueden leer y escribir ocho bits a la vez.

El siguiente paso importante en la evolución de los microprocesadores fue la introducción en 1972 del Intel 8008. Este fue el primer microprocesador de ocho bits y era casi dos veces más complejo que el 4004.

Ninguno de estos pasos tuvo el impacto del siguiente acontecimiento importante: la introducción del Intel 8080 en 1974. Este fue el primer microprocesador de uso general. Mientras que el 4004 y el 8008 habían sido diseñados para aplicaciones específicas, el 8080 fue diseñado para ser la CPU de un microcomputador de propósito general. Al igual que el 8008, el 8080 es un microprocesador de ocho bits. El 8080, sin embargo, es más rápido, tiene un conjunto de instrucciones más rico y tiene una capacidad de direccionamiento mayor.

Sobre la misma época empezaron a desarrollarse los microprocesadores de 16 bits. Sin embargo, hasta el final de los setenta no aparecieron estos potentes microprocesadres de 16 bits de propósito general. Uno de estos fue el 8086. El siguiente paso en esta tendencia ocurrió en 1981, cuando los Laboratorios Bell y Hewlett-Packard desarrollaron microprocesadores de un solo chip de 32 bits. Intel introdujo su microprocesador de 32 bits, el 80386, en 1985 (Tabla 2.6).

Tabla 2.6. Evolución de los microprocesadores Intel.

a) Procesadores de los años setenta

	4004	**8008**	**8080**	**8086**	**8088**
Fecha de introducción	1971	1972	1974	1978	1979
Velocidad de reloj	108 kHz	108 kHz	2 MHz	5 MHz, 8 MHz, 10 MHz	5 MHz, 8 MHz
Ancho del bus	4 bits	8 bits	8 bits	16 bits	8 bits
N.º de transistores	2.300	3.500	6.500	29.000	29.000
Tamaño (µm)	10	—	6	3	6
Memoria direccionable	640 Bytes	16 KB	64 KB	1 MB	1 MB
Memoria virtual	—	—	—	—	—

Tabla 2.6. Evolución de los microprocesadores Intel (continuación).

b) Procesadores de los años ochenta

	80286	386TM DX	386TM SX	486TM DX CPU
Fecha de introducción	1982	1985	1988	1989
Velocidad de reloj	6-12,5 MHz	16-33 MHz	16-33 MHz	25-50 MHz
Ancho del bus	16 bits	32 bits	16 bits	32 bits
N.º de transistores	134.000	275.000	275.000	1,2 millones
Tamaño (µm)	1,5	1	1	0,8-1
Memoria direccionable	16 megabytes	4 gigabytes	16 megabytes	4 gigabytes
Memoria virtual	1 gigabyte	64 terabytes	64 terabytes	64 terabytes

c) Procesadores de los años noventa

	486TM SX	Pentium	Pentium Pro	Pentium II
Fecha de introducción	1991	1993	1995	1997
Velocidad de reloj	16-33 MHz	60-166 MHz	150-200 MHz	200-300 MHz
Ancho del bus	32 bits	32 bits	64 bits	64 bits
N.º de transistores	1,185 millones	3,1 millones	5,5 millones	7,5 millones
Tamaño (µm)	1	0,8	0,6	0,35
Memoria direccionable	4 gigabytes	4 gigabytes	64 gigabytes	64 gigabytes
Memoria virtual	64 terabytes	64 terabytes	64 terabytes	64 terabytes

d) Procesadores recientes

	Pentium III	Pentium 4	Itanium	Itanium II
Fecha de introducción	1999	2000	2001	2002
Velocidad de reloj	450-660 MHz	1,3-1,8 GHz	733-800 MHz	900 MHz-1 GHz
Ancho del bus	64 bits	64 bits	64 bits	64 bits
N.º de transistores	9,5 millones	42 millones	25 millones	220 millones
Tamaño (µm)	0,25	0,18	0,18	0,18
Memoria direccionable	64 gigabytes	64 gigabytes	64 gigabytes	64 gigabytes
Memoria virtual	64 terabytes	64 terabytes	64 terabytes	64 terabytes

Fuente: Intel Corp. **http://www.intel.com/intel/museum/**

2.2. DISEÑO BUSCANDO MEJORES PRESTACIONES

Año tras año, el precio de los computadores continúa cayendo dramáticamente, mientras que las prestaciones y la capacidad de estos sistemas sigue creciendo. En una tienda se puede conseguir un computador, por menos de 1000 dólares, con muchas mejores prestaciones que un IBM de hace diez años. Dentro de un computador personal, incluyendo el microprocesador y la memoria, y otros chips, se pueden conseguir unos cien millones de transistores. No se pueden comprar cien millones de nada por tan poco dinero. Esa cantidad en papel higiénico costaría más de 100 000 dólares.

Por tanto, tenemos prácticamente la potencia del computador «gratis». Esta continua revolución tecnológica, ha habilitado el desarrollo de una sorprendente complejidad y potencia. Por ejemplo, las aplicaciones de oficina que requieren la mayor potencia de los sistemas de hoy en día basados en microprocesadores incluyen:

- Procesamiento de imágenes.
- Reconocimiento del habla.
- Vídeo conferencias.
- Aplicaciones multimedia.
- Almacenamiento de ficheros de voz y vídeo.
- Modelado de simulaciones.

Las estaciones de trabajo ahora soportan aplicaciones de ingeniería y ciencia altamente sofisticadas, así como simulaciones, y pueden aplicar los principios de trabajo en grupo a aplicaciones de imagen y vídeo. Además, en los gestión se está confiando en la creciente potencia de los servidores para manejar transacciones y procesamiento de bases de datos y para soportar redes cliente-servidor masivas que han reemplazado los gigantescos centros de computadores de antaño.

Lo que es fascinante sobre todo desde la perspectiva de la organización y arquitectura de computadores es que, por una parte, los bloques básicos de los portentosos computadores de hoy en día son prácticamente los mismos que los del computador IAS de hace casi cincuenta años, mientras que por otra parte, las técnicas para sacar hasta la última gota del rendimiento de los elementos disponibles se han vuelto cada vez más sofisticadas.

Esta observación sirve de guía principal para la presentación de este libro. A medida que avanzamos en los distintos elementos y componentes de un computador, se persiguen dos objetivos. Primero, el libro explica la funcionalidad fundamental en cada área que se considera, y segundo, el libro explora las técnicas requeridas para conseguir el máximo de prestaciones. En el resto de esta sección, destacamos algunos de los factores que hay tras la necesidad de diseñar para obtener mejores prestaciones.

VELOCIDAD DEL MICROPROCESADOR

Lo que le da al Pentium o al PowerPC esa increíble potencia es la persecución sin descanso del incremento de velocidad por parte de los fabricantes del procesador. La evolución de estas máquinas

continúa confirmando la ley de Moore, mencionada previamente. Siempre y cuando esta ley se cumpla, los fabricantes de chips pueden crear una nueva generación de chips cada tres años (con hasta cuatro veces más de transistores). En los chips de memoria, se ha cuadriplicado cada tres años la capacidad de las memorias dinámicas de acceso aleatorio (DRAM) y esta sigue siendo la tecnología básica de la memoria principal de los computadores actuales. En microprocesadores, la adición de nuevos circuitos, y la potenciación de la velocidad que proviene de la reducción de las distancias entre ellos, ha conseguido cuadruplicar o quintuplicar las prestaciones cada tres años desde que Intel lanzó su familia X86 en 1978.

Pero la velocidad bruta del procesador no alcanzará su potencial al menos que se le alimente con un flujo constante de trabajo en forma de instrucciones. Cualquier cosa que se interponga en el camino de este flujo limita la potencia del procesador. Conforme a esto, mientras que los fabricantes de chips han estado ocupados aprendiendo cómo se fabrican chips de densidad cada vez mayor, los diseñadores del procesador tienen que producir técnicas cada vez más elaboradas para alimentar al «monstruo». Entre las técnicas incorporadas a los procesadores de hoy en día están:

- **Predicción de ramificación:** el procesador se anticipa al software y predice qué ramas o grupos de instrucciones se van a procesar después con mayor probabilidad. Si el procesador acierta la mayoría de las veces, puede precaptar las instrucciones correctas y almacenarlas para mantener al procesador ocupado. Los ejemplos más sofisticados de esta estrategia predicen no solo la siguiente rama sino varias de ellas. Por tanto, la predicción de ramificación incrementa la cantidad de trabajo disponible que el procesador debe de ejecutar.
- **Análisis del flujo de datos:** el procesador analiza qué instrucciones dependen de los resultados de otras instrucciones, o datos, para crear una organización optimizada de instrucciones. De hecho, las instrucciones se planifican para ser ejecutadas cuando estén listas, independientemente del orden original del programa. Esto evita retrasos innecesarios.
- **Ejecución especulativa:** utilizando la predicción de ramificación y el análisis de flujo de datos, algunos procesadores ejecutan especulativamente instrucciones antes de que aparezcan en la ejecución del programa, manteniendo los resultados en posiciones temporales. Esto permite al procesador mantener sus elementos de ejecución tan ocupados como sea posible ejecutando anticipadamente instrucciones que es probable que se necesiten.

Estas y otras sofisticadas técnicas se hacen necesarias simplemente porque el procesador es muy potente, haciendo así posible explotar la potencia bruta del procesador.

EQUILIBRIO DE PRESTACIONES

Mientras que la velocidad del procesador ha crecido con increíble rapidez, otros componentes esenciales del computador no lo han hecho tan rápido. El resultado de esto es que ahora hace falta prestar atención al equilibrio de las prestaciones: ajustar la organización y la arquitectura para compensar las desigualdades de capacidad entre los distintos componentes.

El problema creado por tales desigualdades no es de ninguna manera más grave que en la interfaz entre el procesador y la memoria principal. Observemos la historia representada en la Figura 2.10. Mientras la velocidad del procesador y la capacidad de la memoria han crecido rápidamente, la velocidad con la que los datos pueden ser transferidos entre la memoria principal y el procesador se ha

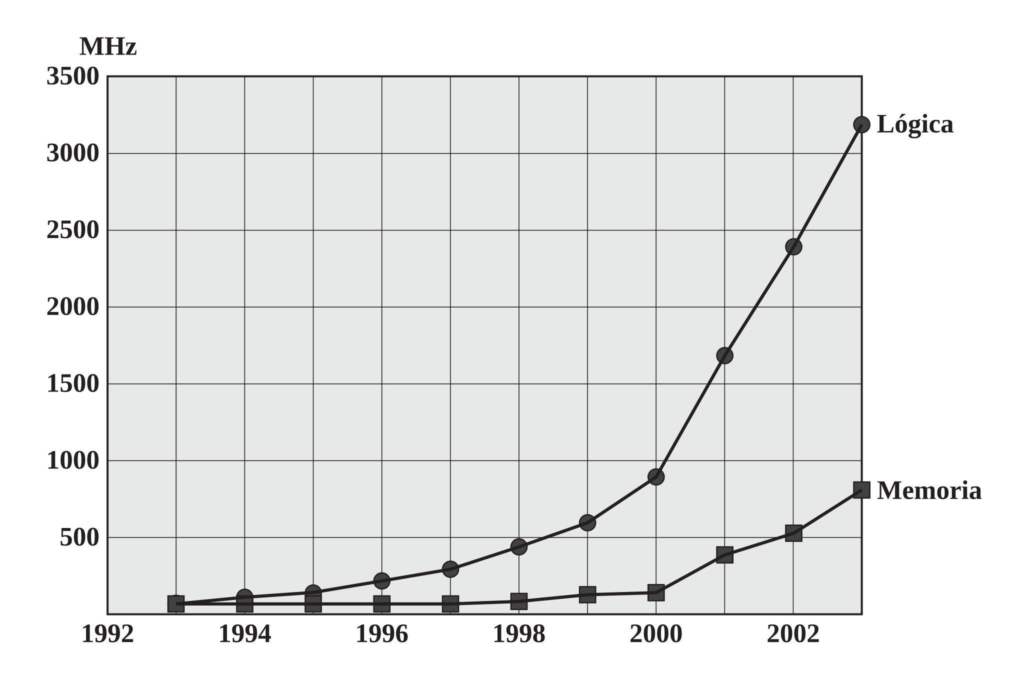


Figura 2.10. Prestaciones de la memoria y la lógica [BORK03].

quedado dramáticamente retrasada. La interfaz entre el procesador y la memoria principal es el camino más importante de todo el computador, ya que es el responsable de llevar el constante flujo de instrucciones y datos entre los chips de la memoria y el procesador. Si la memoria o la interfaz no logran mantener el ritmo de las insistentes demandas del procesador, este se estanca en una posición de espera y se pierde así tiempo de procesamiento valioso.

Hay varias maneras de que una arquitectura pueda atacar este problema, y todas se reflejan en los diseños de computadores contemporáneos. He aquí algunos ejemplos:

- Incrementar el número de bits que se recuperan de una sola vez haciendo las DRAM más «anchas» en lugar de más «profundas» utilizando buses de datos más anchos.
- Cambiar la interfaz DRAM para hacerla más eficiente, incluyendo una caché [6] u otro esquema de almacenamiento temporal en el chip DRAM.
- Reducir la frecuencia del acceso a memoria incorporando, entre el procesador y la memoria principal, unas caché cada vez más complejas y eficientes. Esto incluye la incorporación de una o más caches en el chip del procesador así como una caché fuera del chip cerca del procesador.
- Incrementar el ancho de banda entre el procesador y la memoria usando buses de más alta velocidad y una jerarquía de buses para almacenar y estructurar el flujo de datos.

[6] Una caché es un memoria rápida relativamente pequeña colocada entre una memoria mayor y más lenta y la lógica que accede a la memoria grande. La caché mantiene los datos recientemente accedidos y se diseña para acelerar los sucesivos accesos a los mismos datos. Las cachés se estudian en el Capítulo 4.

Otra área de diseño se centra en el manejo de dispositivos de E/S. Conforme los computadores se hacen más rápidos y potentes, se desarrollan aplicaciones más sofisticadas que se apoyan en el uso de periféricos con demandas intensivas de E/S. La Figura 2.11 muestra algunos ejemplos de los dispositivos periféricos típicos que se usan en ordenadores personales y estaciones de trabajo. Estos dispositivos crean una tremenda demanda de procesamiento de datos. La generación actual de procesadores puede manejar los datos producidos por esos dispositivos, pero aún queda el problema de mover esos datos entre el procesador y los periféricos. Las estrategias en esto incluyen esquemas de cachés y almacenamiento además del uso de buses de interconexión de más alta velocidad y con estructuras más elaboradas. También el uso de configuraciones multiprocesador puede ayudar a satisfacer las demandas de E/S.

La clave en todo esto es el equilibrio. Los diseñadores luchan constantemente por alcanzar el equilibrio en la demanda de rendimiento y procesamiento por parte de los componentes del procesador, la memoria principal, los dispositivos de E/S y de las estructuras de interconexión. Y este diseño tiene que ser constantemente replanteado para hacer frente a dos factores en continua evolución:

- La velocidad a la que el rendimiento está cambiando en las distintas áreas tecnológicas (procesador, buses, memoria, periféricos) difiere enormemente de un tipo de elemento a otro.
- Las nuevas aplicaciones y nuevos dispositivos periféricos cambian constantemente la naturaleza de la demanda en el sistema en cuanto al perfil de instrucción típico y el modelo de acceso de datos.

Así, el diseño de computadores es una forma de arte en constante evolución. Este libro intenta plantear los fundamentos en los que se basa esta forma de arte y dar una visión general de su estado actual.

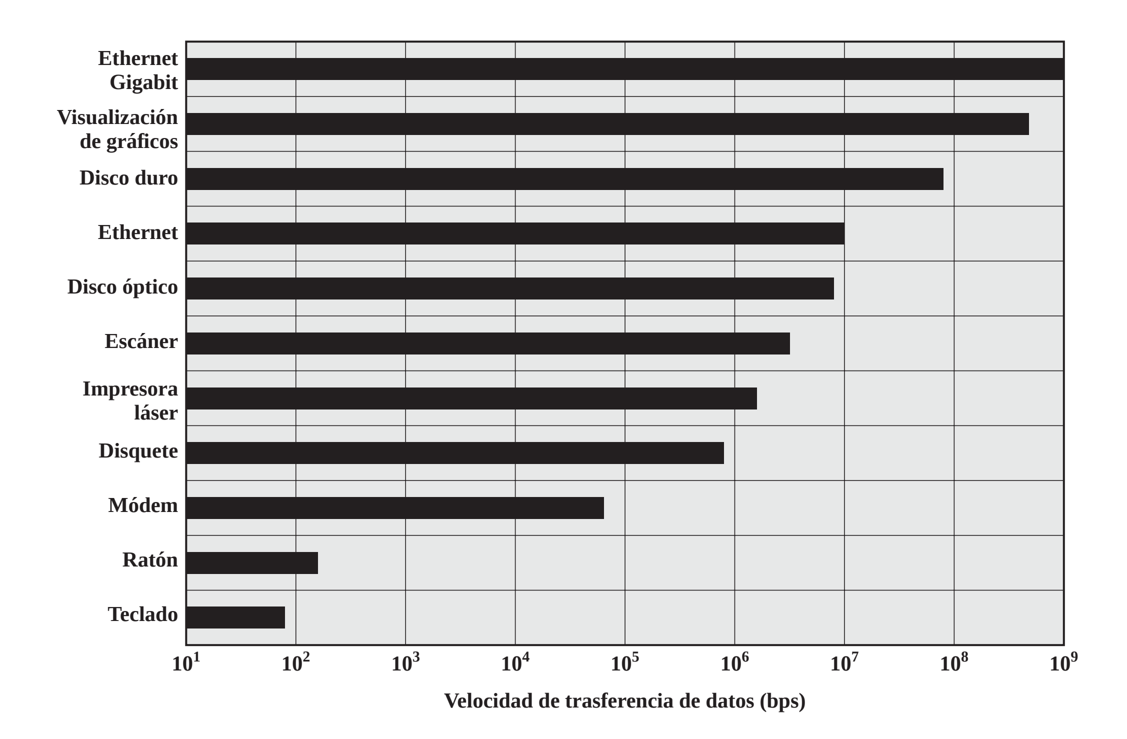


Figura 2.11. Velocidades de transmisión de datos en E/S típicas.

MEJORAS EN LA ORGANIZACIÓN Y ARQUITECTURA DE CHIPS

Además del reto de los diseñadores de equilibrar las prestaciones del procesador con la memoria principal y con otros componentes del computador, persiste la necesidad de aumentar la velocidad del procesador. Hay tres formas de conseguir incrementar la velocidad del procesador:

- Incrementando la velocidad del hardware del procesador: esto se consigue fundamentalmente disminuyendo el tamaño de las puertas lógicas del chip del procesador, de forma que se puedan encapsular más puertas, más cercanas y así incrementar la frecuencia del reloj. Con las puertas más juntas, el tiempo de propagación de las señales se reduce significativamente, haciendo posible un aumento de la velocidad del procesador. Un incremento en la velocidad del reloj implica que las operaciones individuales se ejecuten más rápidamente.
- Incrementando el tamaño y la velocidad de las cachés interpuestas entre el procesador y la memoria principal. En particular, dedicando una parte del chip del procesador a la caché, el tiempo de acceso de la caché disminuye considerablemente.
- Haciendo cambios en la organización y arquitectura del procesador de forma que se incremente la velocidad efectiva de la ejecución de una instrucción. Usualmente, esto implica utilizar paralelismo de una forma u otra.

Tradicionalmente, el factor dominante en la ganancia de prestaciones se debe al aumento de la velocidad del reloj y a la densidad de la lógica. En la Figura 2.12 se muestra esta tendencia en los procesadores de Intel. Sin embargo, al aumentar la velocidad de reloj y la densidad de la lógica, una serie de obstáculos se hacen más significativos [INTE04b]:

- **Potencia**: a medida que la densidad de la lógica y la velocidad del reloj de un chip aumentan, también lo hace la densidad de potencia (watios/cm^2). La dificultad en disipar el calor generado por la gran densidad y la alta velocidad en el chip se convierte en un problema de diseño serio ([GIBB04], [BORK03]).
- **Retardo RC**: la velocidad a la que los electrones pueden fluir en un chip entre transistores está limitada por la resistencia y capacidad de los hilos metálicos que los conectan; concretamente, el retardo aumenta al hacerlo el producto RC. Como los componentes del chip disminuyen de tamaño, los hilos que los interconectan son más finos, aumentando su resistencia. Además, los hilos están más juntos, aumentando la capacidad.
- **Latencia de memoria**: la velocidad de la memoria ralentiza la velocidad del procesador, como se ha visto anteriormente.

Por tanto, es necesario hacer más énfasis en la organización y arquitectura para mejorar las prestaciones. La Figura 2.12 destaca los principales cambios que se han hecho a lo largo de los años para aumentar el paralelismo y, por tanto, la eficiencia computacional de los procesadores. Estas técnicas se verán en capítulos posteriores de este libro.

Empezando por finales de los ochenta, y siguiendo quince años más, se utilizaban principalmente dos estrategias para aumentar las prestaciones, más allá de lo que se puede conseguir sencillamente incrementando la velocidad del reloj. Primero, hubo un incremento de la capacidad de la caché. Ahora hay usualmente dos o tres niveles de caché entre el procesador y la memoria principal. Al aumentar la densidad del chip, se ha incorporado más memoria caché en el chip, haciendo posible un acceso más

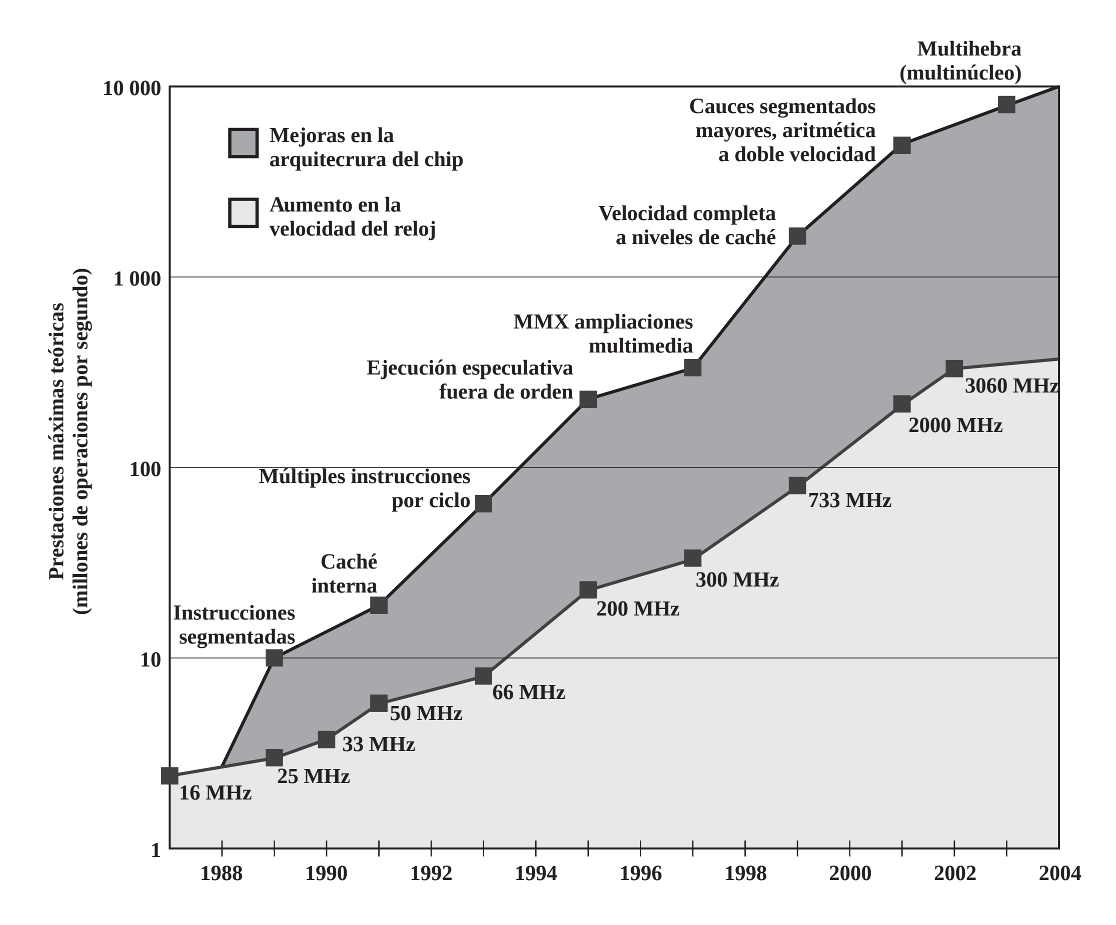


Figura 2.12. Prestaciones de los microprocesadores Intel [GIBB04].

rápido a la caché. Por ejemplo, el chip original Pentium cedía alrededor de un diez por ciento del área del chip para la caché. El más reciente Pentium 4 deja la mitad del área del chip para las cachés.

Segundo, la lógica de ejecución de una instrucción dentro de un procesador está siendo cada vez más compleja para posibilitar la ejecución paralela de instrucciones dentro del procesador. Dos enfoques de diseño notables han sido la segmentación de cauce y el superescalado. Una segmentación de cauce funciona como una línea de ensamblado de una planta industrial, haciendo posible que se ejecuten a la vez varias etapas de diferentes instrucciones a lo largo del cauce. La aproximación superescalar permite, en esencia, varios cauces dentro de un único procesador, de forma que instrucciones que no dependan unas de otras se puedan ejecutar en paralelo.

Ambas aproximaciones están alcanzando un punto que no da más de sí. La organización interna de los procesadores actuales es excesivamente compleja y se puede conseguir mucho paralelismo fuera del conjunto de instrucciones. Parece probable que otras mejoras significativas en esta dirección serán relativamente modestas [GIBB04]. Con tres niveles de caché en el chip del procesador, proporcionando cada nivel capacidad sustancial, también parece que los beneficios de la caché están alcanzando un límite.

Sin embargo, fiarse simplemente del aumento de la velocidad de reloj para aumentar las prestaciones lleva al problema de la disipación de potencia que ya hemos mencionado. Cuanto mayor sea la velocidad del reloj, mayor será la potencia que hay que disipar, y se pueden alcanzar algunos límites físicos básicos.

Con todas estas dificultades en mente, los diseñadores han recurrido a una aproximación fundamentalmente nueva: situar varios procesadores en el mismo chip, con una gran caché compartida. El uso de varios procesadores en el mismo chip, también conocido como varios núcleos, da potencia para incrementar las prestaciones sin aumentar la velocidad del reloj. Hay estudios que indican que, dentro de un procesador, el aumento de las prestaciones es aproximadamente proporcional a la raíz cuadrada del incremento de la complejidad [BORK03]. Pero si el software puede soportar el uso efectivo de varios procesadores, entonces duplicando el número de procesadores casi se duplican las prestaciones. Por tanto, la estrategia es usar dos simples procesadores en el chip en lugar de un procesador más complejo.

Además, con dos procesadores se pueden justificar grandes cachés. Esto es importante porque el consumo de potencia de la lógica de la memoria en un chip es mucho menor que la de la lógica de un procesador. Podemos esperar, en próximos años, que la mayoría de los nuevos chips de procesadores tendrán varios procesadores [ANTH04].

El primer chip multiprocesador comercialmente disponible se introdujo en el 2001, y era el chip POWER4 de IBM, que se basaba en la misma organización y arquitectura usada en los chips del PowerPC. La Figura 2.13, basada en una figura de [TEND02], da una vista en forma de niveles de la arquitectura del chip.

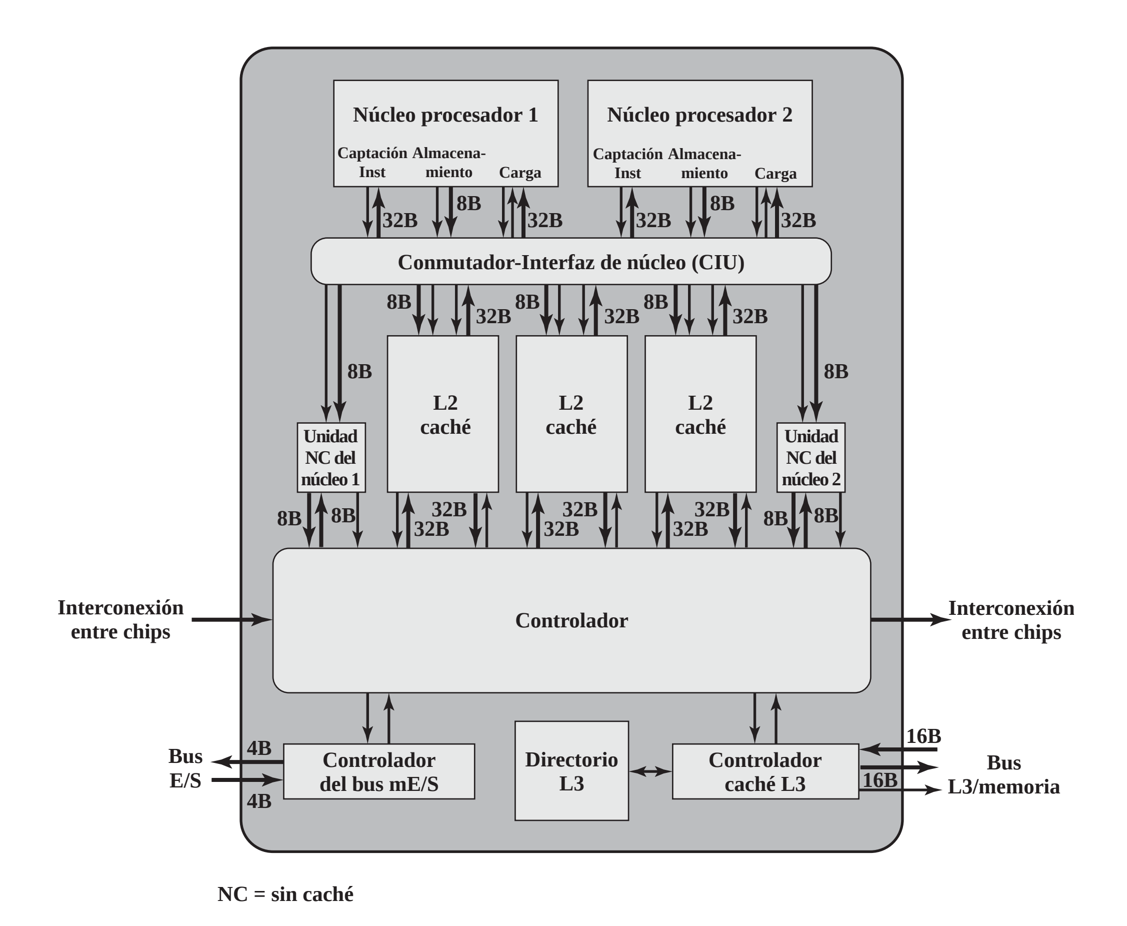


Figura 2.13. Organización del chip POWER4.

2.3. EVOLUCIÓN DEL PENTIUM Y DEL POWERPC

A través de este libro nos basamos en muchos ejemplos concretos de diseño e implementación de computadores para ilustrar conceptos y aclarar los compromisos. El libro se basa la mayor parte de las veces en ejemplos de dos familias de computadores: el Pentium de Intel y el PowerPC. El Pentium representa el resultado de décadas de esfuerzo de diseño en computadores de repertorio complejo de instrucciones (CISC). Incorpora los sofisticados principios de diseño que antes se encontraban solo en ordenadores grandes y supercomputadores, y es un excelente ejemplo de diseño CISC. El PowerPC es descendiente directo del primer sistema RISC, el IBM 801, y es uno de los sistemas basados en RISC más potentes y mejor diseñados del mercado.

En esta sección damos una breve visión general de ambos sistemas.

PENTIUM

Intel ha sido el número uno de los fabricantes de microprocesadores durante décadas, una posición que no parece probable que abandone. La evolución de su microprocesador más representativo es un buen indicador de la evolución de la tecnología de computadores en general.

La Tabla 2.6 muestra esta evolución. Es interesante que, como los microprocesadores han crecido rápidamente y de forma mucho más compleja, Intel ha sabido aprovechar la oportunidad. Intel solía desarrollar un microprocesador cada cuatro años. Pero Intel espera dejar sus rivales atrás reduciendo este tiempo de desarrollo a un año o dos, y así lo ha hecho con sus generaciones recientes de Pentium.

Merece la pena enumerar algunos de los rasgos más destacados de la evolución de los productos Intel:

- **8080:** es el primer microprocesador de propósito general del mundo. Era una máquina de ocho bits, con datos de memoria de ocho bits. El 8080 se uso en el primer computador personal, el Altair.
- **8086:** un circuito de 16 bits, mucho más potente. Además de un camino de datos más ancho y registros más grandes, el 8086 tenía una caché de instrucción, o cola, que precaptaba algunas instrucciones antes de ser ejecutadas. Una variante de este procesador, el 8088, se utilizó en el primer computador personal de IBM, asegurando el éxito de Intel.
- **80286:** esta ampliación del 8086 permitía direccionar una memoria de 16 MB en lugar de solo 1 MB.
- **80386:** fue la primera máquina de Intel con 32 bits, y constituyó una gran revisión del modelo anterior. Con una arquitectura de 32 bits, el 80386 rivalizaba en complejidad y potencia con los minicomputadores y grandes computadores introducidos en el mercado pocos años antes. Este fue el primer procesador de Intel que admitió multitarea, significando esto que podría ejecutar varios programas a la vez.
- **80486:** el 80486 introduce el uso de tecnología de caché mucho más sofisticada y potente e instrucciones de segmentación de cauce sofisticadas. El 80486 también tenía un coprocesador matemático, descargando a la CPU principal de las operaciones matemáticas complejas.
- **Pentium:** con el Pentium, Intel introduce el uso de técnicas superescalares que permiten que varias instrucciones se ejecuten en paralelo.

- **Pentium Pro:** el Pentium Pro continuó la tendencia iniciada con el Pentium hacia la organización superescalar, con el uso agresivo del renombrado de registros, predicción de ramificaciones, análisis del flujo de datos, y ejecución especulativa.
- **Pentium II:** en el Pentium II se incorporó la tecnología Intel MMX, que se diseñó específicamente para procesar de forma eficiente datos de vídeo, audio y gráficos.
- **Pentium III:** el Pentium III incorpora instrucciones adicionales en coma flotante para procesar software de gráficos 3D.
- **Pentium 4:** el Pentium 4 incluye coma flotante adicional y otras mejoras multimedia[7].
- **Itanium:** esta nueva generación de procesadores Intel usan una organización de 64 bits con arquitectura IA-64, que se verá con detalle en el Capítulo 15.
- **Itanium 2:** esta actualización del Itanium original incluye una serie de mejoras en el hardware para aumentar la velocidad.

POWERPC

En 1975, el proyecto minicomputador 801 de IBM fue el primero en muchos de los conceptos de arquitectura usados en sistemas RISC. El 801, junto con el procesador RISC I de Berkeley, inició el movimiento RISC. El 801, sin embargo, era simplemente un prototipo que intentaba demostrar nuevos conceptos de diseño. El éxito del proyecto 801 permitió a IBM desarrollar una estación de trabajo RISC comercial, el RT PC. El RT PC, introducido en 1986, adaptaba los conceptos arquitectónicos del 801 a un producto real. El RT PC no tuvo éxito comercial y tuvo muchos rivales con prestaciones comparables o mejores. En 1990, IBM produjo un tercer sistema que incorporaba las lecciones aprendidas con el 801 y el RT PC. El IBM RISC System/6000 era una máquina superescalar RISC comercializada como una estación de trabajo de altas prestaciones; poco después de su introducción, IBM comenzó a llamarla arquitectura POWER.

Como siguiente paso, IBM se alió con Motorola, que había desarrollado las series 68000 de microprocesadores, y Apple, que usaba el chip de Motorola en sus computadores Macintosh. El resultado es una serie de máquinas que implementan la arquitectura PowerPC. Esta arquitectura deriva de la arquitectura Power (Figura 2.13). Se hicieron cambios para añadir características clave que no estaban y para permitir una implementación más eficiente eliminando algunas instrucciones y relajando la especificación para eliminar casos especialmente problemáticos. La arquitectura PowerPC resultante es un sistema RISC superescalar.

Hasta ahora se han presentado cuatro miembros de la familia PowerPC (Tabla 2.7):

- **601:** el propósito del 601 era llevar la arquitectura PowerPC al mercado lo más rápidamente posible. El 601 es una máquina de 32 bits.
- **603:** pensado para computadores de aplicación final de sobremesa y portátiles. Es también una máquina de 32 bits, comparable en prestaciones con el 601, pero con coste más bajo e implementación más eficiente.

---

[7] Con el Pentium 4, Intel cambió de la numeración romana a la arábiga para designar los números de la serie.

Tabla 2.7. Resumen de los procesadores PowerPC.

	Año de comercialización	Velocidad de reloj	Cache L1	Cache L1	Número de transistores (10^6)
601	1993	50-120 MHz	—	—	2,8
603/603e	1994	100-300 MHz	16 KB instr 16 KB data	—	1,6-2,6
604/604e	1994	166-350 MHz	32 KB instr 32 KB data	—	3,6-5,1
740/750 (G3)	1997	200-366 MHz	32 KB instr 32 KB data	256 KB- 1 MB	6,35
G4	1999	500 MHz	32 KB instr 32 KB data	256 KB- 1 MB	—
G5	2003	2,5 GHz	64 KB instr 32 KB data	512 KB	58

- **604:** pensado para computadores de aplicación final de sobremesa y portátiles. De nuevo es una máquina de 32 bits pero utiliza técnicas de diseño superescalares, mucho más avanzadas para lograr mayores prestaciones.
- **620:** pensado para servidores de aplicación final. El primer miembro de la familia PowerPC que implementa una arquitectura completa de 64 bits, incluyendo registros y buses de datos de 64 bits.
- **740/750:** también conocido como procesador G3. Este procesador integra dos niveles de caché en el chip del procesador principal, ofreciendo mejoras significativas en las prestaciones sobre otras máquinas comparables con organización de cache fuera del chip.
- **G4:** este procesador continúa incrementando el paralelismo y la velocidad interna del chip del procesador.
- **G5:** introduce mejoras adicionales en el paralelismo, en la velocidad interna y constituye una organización de 64 bits.

2.4. LECTURAS Y SITIOS WEB RECOMENDADOS

Se puede encontrar una descripción de la serie IBM 7000 en [BELL71a]. Hay un buen tratamiento del IBM 360 en [SIEW82] y del PDP-8 y otras máquinas DEC en [BELL78a]. Estos tres libros también contienen numerosos y detallados ejemplos de otros computadores que abarcan la historia de los computadores de principios de los años ochenta. Un libro más reciente que incluye un excelente conjunto de estudios sobre máquinas históricas se encuentra en [BLAA97]. En [BETK97] se incluye una buena historia de los microprocesadores.

[OLUK96], [HAMM97] y [SAKA02] tratan sobre la utilización de varios procesadores en un solo chip.

Uno de los mejores tratamientos del Pentium se encuentra en [SHAN05]. La propia documentación de Intel también es buena [INTE04]. [BREY03] proporciona una buena visión de la línea de microprocesadores Intel. Sobre la arquitectura Itanium, *véase* [INTE02] y [EVAN03].

Véase [HUTC96], [SCHA97] y [BOHR98] para discusiones interesantes sobre la ley de Moore y sus consecuencias.

BELL71a Bell, C. y Newell, A.: *Computer Structures: Readings and Examples.* New York. McgGraw-Hill, 1971.

BELL78a Bell, C.; Mudge, J. y McNamara, J.: *Computer Engineering: A DEC View of Hardware Systems Design.* Bedford, MA: Digital Press, 1978.

BETK97 Betker, M.; Fernando, J. y Whalen, S.: «The History of the Microprocessor». *Bell Labs Technical Journal,* otoño, 1997.

BLAA97 Blaauw, G. y Brooks, F.: *Computer Architecture: Concepts and Evolution.* Reading, MA. Addison-Wesley, 1997.

BOHR98 Bohr, M.: «Silicon Trends and Limits for Advanced Microprocessors». *Communications of the ACM,* marzo, 1998.

BREY03 Brey, B.: *The Intel Microprocessors: 8086/8066, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4.* Upper Saddle River, NJ. Prentice Hall, 2000.

EVAN03 Evans, J. y Trimper, G.: *Itanium Architecture for Programmers.* Upper Saddle River, NJ. Prentice Hall, 2003.

HAMM97 Hammond, L.; Nayfay, B. y Olukotun, K.: «A Single-Chip Multiprocessor». *Computer,* septiembre, 1997.

HUTC96 Hutcheson, G. y Hutcheson, J.: «Technology and Economics in the Semiconductor Industry». *Scientific American,* enero, 1996.

INTE02 Intel Corp.: *Intel Itanium Architecture Software Developer's Manual (3 volúmenes).* Aurora, CO, 2002.

INTE04a Intel Corp.: *IA-32 Intel Architecture Software Developer's Manual (3 volúmenes).* Aurora, CO, 2004.

OLUK96 OLUKOTUN, K., *et al.*: «The Case for a Single-Chip Multiprocessor». *Proceedings, Seventh International Conference on Architectural Support for Programming Languages and Operating Systems,* 1996.

SAKA02 Sakai, S.: «CMP on SoC. architect's view». Proccedings. 15th international symposium on System Synthesis, 2002.

SCHA97 Schaller, R.: «Moore's Law: Past, Present, and Future». *IEEE Spectrum,* junio, 1997.

SHAN05 Shanley, T.: *Unabridged Pentium 4, The: IA32 Processor Genealogy.* Reading, MA. Addison-Wesley, 2005.

SIEW82 Siewiorek, D.; Bell, C. y Newell, A.: *Computer Structures: Principles and Examples.* New York. McGraw-Hill, 1982.

SITIOS WEB RECOMENDADOS

- **Intel developer's page:** página web de desarrollos de Intel; es un punto de inicio para acceder a la información sobre Pentium. También incluye la revista de tecnología de Intel (*Intel Technology Journal*).
- **PowerPC:** dos sitios web sobre el PowerPC, una de IBM y otra de la empresa Freescale Semiconductor, formalmente sector de semiconductores de Motorola.

- **Standard Performance Evaluation Corporation (empresa de estándares de evaluación de prestaciones):** SPEC es una organización ampliamente reconocida en la industria de los computadores por sus desarrollos en estándares de programas de pruebas (*benchmarks*) utilizados para medir y comparar las prestaciones de distintos computadores.
- **El Top500 de los supercomputadores:** contiene una breve descripción de la arquitectura y organización de los supercomputadores actuales, y además comparaciones.
- **Instituto Charles Babbage:** proporciona enlaces a numerosos sitios web sobre la historia de los computadores.

2.5. PALABRAS CLAVE, PREGUNTAS DE REPASO Y PROBLEMAS

PALABRAS CLAVE

acumulador (AC)	registro de instrucción (IR)	código de operación
unidad aritmético lógica (ALU)	conjunto de instrucciones	fabricante de equipos originales (OEM)
chip	circuito integrado (IC)	unidad de control de programa
canal de datos	memoria principal	contador de programa (PC)
fase de ejecución	registro de dirección de memoria (MAR)	programa cargado
fase de captación	registro buffer de memoria (MBR)	compatibilidad hacia arriba.
entrada/Salida	microprcesador	máquina von Neumann
registro buffer de instrucción (IR)	multiplexor	oblea
ciclo de instrucción		palabra

PREGUNTAS DE REPASO

2.1. ¿Qué es un computador de programa almacenado?
2.2. ¿Cuáles son los cuatro componentes principales de un computador de uso general?
2.3. A nivel de circuito integrado, ¿cuáles son los tres componentes principales de un computador?
2.4. Explicar la ley de Moore.
2.5. Enumerar las características clave de una familia de computadores.
2.6. ¿Cuál es la clave para distinguir las características de un microprocesador?

PROBLEMAS

2.1. Sean A = A(1), A(2)..., A(1000) y B = B(1), B(2)..., B(1000) dos vectores (unidimensionales) que comprenden 1000 números cada uno, que van a ser sumados para formar un vector C, tal que C(I) = A(I) + B(I), donde I = 1, 2..., 1000. Usando el conjunto de instrucciones IAS, escribir un programa para resolver este problema.

2.2. *a)* En el IAS, ¿cómo sería el código de la instrucción máquina para cargar el contenido de memoria de la dirección 2?

b) ¿Cuántos accesos a memoria tendría que hacer la CPU para completar esta instrucción durante el ciclo instrucción?

2.3. En el IAS, describir con palabras el proceso que tiene que seguir la CPU para leer un valor de memoria y escribir un valor en memoria, indicando cómo cambian MAR, MBR, el bus de direcciones, el bus de datos y el bus de control.

2.4. Dados los contenidos de memoria siguientes de un computador IAS:

Direcciones	Contenidos
08A	010FA210FB
08B	010FA0F08D
08C	020FA210FB

Mostrar el código del programa en lenguaje ensamblador, empezando en la dirección 08A. Explicar lo que hace el programa.

2.5. Indicar, en la Figura 2.3, el ancho en bits de cada camino de datos (por ejemplo, entre el AC y la ALU).

2.6. En el IBM 360 modelos 65 y 75, las direcciones están situadas en dos unidades de memoria principal separadas (por ejemplo, todas las palabras pares en una unidad y todas las impares en otra). ¿Cuál puede ser el objetivo de esta técnica?

2.7. En la Tabla 2.4 se puede ver que las prestaciones relativas del Modelo 75 del IBM 360 son cincuenta veces las del Modelo 30, mientras que el tiempo de ciclo de instrucción es solo cinco veces más rápido. ¿Cómo se explica esta discrepancia?

2.8. En la tienda de computadores de Miguel Ángel Valenzuela, escuchas a un cliente preguntando por el computador más rápido de la tienda que pueda comprar. Miguel Ángel Valenzuela le contesta, «estás mirando nuestros Macintosh. El Mac más rápido que tenemos tiene una velocidad de reloj de 1,2 Gigahercios. Si realmente quieres una máquina rápida, debes comprarte un Intel Pentium IV a 2,4 Gigahercios». ¿Es correcto lo que dice Miguel Ángel Valenzuela? ¿Qué debería decirle para ayudar a su cliente?

2.9. El ENIAC era una máquina decimal, donde un registro se representaba con un anillo de diez tubos de vacío. En un instante dado, solo un tubo de vacío estaba en estado ON, representando uno de 10 dígitos. Suponiendo que el ENIAC tuviera la capacidad de tener varios tubos de vacío en los estados ON y OFF simultáneamente, ¿por qué esta representación es un «despilfarro» y qué rango de valores enteros se podrían representar con diez tubos de vacío?

2.10. Un procesador está sincronizado por un reloj con una frecuencia constante f o, lo que es lo mismo, un tiempo de ciclo constante τ, donde $\tau = 1/f$. El tamaño de un programa se puede medir con el número de instrucciones máquina, o número de instrucciones I_c, que contiene el programa. Distintas instrucciones máquina consumen distintos ciclos de reloj. Un parámetro importante es el número medio de ciclos por instrucción CPI de un programa. El tiempo T que el procesador necesita para ejecutar un programa dado se puede expresar:

$$T = I_c \times CPI \times \tau$$

Esta fórmula se puede reescribir teniendo en cuenta que durante la ejecución de una instrucción, parte del trabajo lo realiza el procesador, y parte del tiempo se está transfiriendo a, o desde memoria, una palabra. En este último caso, el tiempo de transferencia depende del tiempo de ciclo de memoria, que puede ser mayor que el tiempo de ciclo del procesador. Se puede reescribir la ecuación anterior:

$$T = I_c \times [\, p + (m \times k)] \times \tau$$

donde p es el número de ciclos de procesador necesarios para decodificar y ejecutar la instrucción, m es el número de accesos a memoria, y k es el cociente entre el tiempo de ciclo de memoria y el tiempo de ciclo del procesador. Los cinco factores de prestaciones de la ecuación anterior (I_c, p, m, k, τ) dependen de los cuatro atributos del sistema: el diseño del conjunto de instrucciones (conocido como arquitectura del conjunto de instrucciones), de la tecnología del compilador (cómo de efectivo es el compilador produciendo un programa en lenguaje máquina a partir de un programa en un lenguaje de alto nivel), de la implementación del procesador, y de la jerarquía de memoria y de la caché. Hacer un matriz en la que una dimensión muestre los cinco factores de las prestaciones y la otra los cuatro atributos del sistema. Poner una X en cada celda en la que el atributo del sistema afecte al factor de prestaciones.

2.11. Una medida usual de las prestaciones de un procesador es la velocidad a la que ejecuta instrucciones, expresada en millones de instrucciones por segundo (MIPS). Expresar la velocidad en MIPS en función de la velocidad de reloj y *CPI*.

2.12. Los primeros ejemplos de diseños CISC y RISC son el VAX 11/780 y el IBM RS/6000, respectivamente. Utilizando un programa de pruebas típico, se obtienen los siguientes resultados:

Procesador	Frecuencia de reloj	Tasa de ejecución de instrucciones	Tiempo de la CPU
VAX 11/780	5 MHz	1 MIPS	12 x segundos
IBM RS/6000	25 MHz	18 MIPS	x segundos

La última columna muestra que el VAX necesita doce veces más tiempo de CPU que el IBM.

a) ¿Cuál es el valor relativo del número de instrucciones del código máquina para este programa de prueba en las dos máquinas?

b) ¿Cuáles son los valores CPI de las dos máquinas?

2.13. Un programa de prueba se está ejecutando en un procesador de 40 MHz. El código objeto consta de 100 000 instrucciones, con el siguiente conjunto de instrucciones y ciclo de reloj:

Tipo de Instrucción	Contador de instrucciones	Ciclo de reloj
Aritméticos enteros	45 000	1
Datos transferidos	32 000	2
Punto flotante	15 000	2
Control transferidos	8 000	2

Determinar el CPI efectivo, la velocidad en MIPS, y el tiempo de ejecución para este programa.

2.14. Para obtener una comparación fiable de las prestaciones de varios computadores, es preferible ejecutar diferentes programas de pruebas en cada máquina, y entonces promediar los resultados. Por ejemplo, con m programas diferentes, se puede calcular una simple media aritmética como sigue:

$$R_a = \frac{1}{m} \sum_{i=1}^{m} R_i$$

Donde R_i es la velocidad en MIPS del i-ésimo programa de prueba. Una alternativa es calcular la media armónica:

$$R_h = \frac{m}{\sum_{i=1}^{m} \frac{1}{R_i}}$$

a) Comentar las ventajas y desventajas de ambos métodos. Consejo: considerar el tiempo de ejecución medio (en microsegundos) por instrucción, para el programa i, $T_i = 1/R_i$.

b) Se ejecutan cuatro programas de prueba en tres computadores con los siguientes resultados:

	Computador A	**Computador B**	**Computador C**
Programa 1	1	10	20
Programa 2	1 000	100	20
Programa 3	500	1 000	50
Programa 4	100	800	100

La tabla muestra el tiempo de ejecución en segundos, al ejecutar 100 000 000 de instrucciones en cada uno de los cuatro programas. Calcular las medias aritmética y armónica, y establecer un *ranking* basándose en la media armónica.

PARTE 2

EL COMPUTADOR

CUESTIONES A TRATAR EN LA SEGUNDA PARTE

Un computador consta de procesador, memoria, E/S, y las interconexiones entre estos componentes principales. Con la excepción del procesador, que es suficientemente complejo para dedicar a su estudio toda la Parte Tres, la Parte Dos examina cada uno de estos componentes con detalle.

ESQUEMA DE LA SEGUNDA PARTE

CAPÍTULO 3. PERSPECTIVA DE ALTO NIVEL DEL FUNCIONAMIENTO Y DE LAS INTERCONEXIONES DEL COMPUTADOR

A alto nivel, un computador consta de procesador, memoria, y unidades de E/S. El sistema funciona intercambiando señales de datos y control entre sus componentes. Para permitir dicho intercambio, los componentes deben estar interconectados. El Capítulo 3 comienza con un breve examen de los componentes del computador y sus requisitos de entrada/salida. Después, el capítulo analiza los aspectos clave que afectan el diseño de la estructura de interconexión, especialmente los relativos a las interrupciones. La mayor parte del capítulo se centra en el estudio de la aproximación más usual para interconectar los componentes del computador: las estructuras de buses.

CAPÍTULO 4. MEMORIA CACHÉ

La memoria del computador presenta una amplia variedad de tipos, tecnologías, organizaciones, prestaciones y costes. Un computador típico posee una jerarquía de subsistemas de memoria, incluyendo tanto memoria interna (el procesador puede acceder a ella directamente) como externa (el procesador accede a ella a través de una unidad de E/S). El Capítulo 4 comienza con una revisión de esta jerarquía. A continuación, el capítulo aborda el diseño de la memoria caché, incluyendo las cachés separadas para datos e instrucciones y las cachés de dos niveles.

CAPÍTULO 5. MEMORIA INTERNA

El diseño del sistema de memoria principal constituye una guerra sin cuartel entre tres requisitos de diseño: capacidad de almacenamiento elevada, tiempo de acceso reducido, y bajo coste. A medida que evoluciona la tecnología de las memorias, cada uno de estos aspectos cambia de forma que las decisiones en el diseño de la organización de la memoria principal debe reconsiderarse a medida que surgen nuevas implementaciones. El Capítulo 5 se centra en el diseño de la memoria interna. En primer lugar se examinan la naturaleza y la organización de la memoria principal de semiconductores. Después, se exploran los avances más recientes en el ámbito de la organización de la memoria DRAM.

CAPÍTULO 6. MEMORIA EXTERNA

Para disponer de una capacidad de almacenamiento verdaderamente elevada y conseguir un almacenamiento más duradero que el que proporciona la memoria principal, se necesita la memoria externa. El tipo de memoria externa más ampliamente utilizado es el disco magnético, de forma que la mayor parte del Capítulo 5 se centra en dicho tópico. En primer lugar, se analizan la tecnología de los discos magnéticos y las consideraciones de diseño. A continuación se estudia el uso de la organización RAID para mejorar las prestaciones de la memoria de disco. El Capítulo 6 también examina el almacenamiento óptico y las cintas magnéticas.

CAPÍTULO 7. ENTRADA/SALIDA

Las unidades de E/S están conectadas con el procesador y la memoria principal, y cada una de ellas controla uno o más dispositivos periféricos. El Capítulo 7 se dedica a distintos aspectos de la organización de E/S. Se trata de un área compleja, y mucho menos comprendida que otras áreas del diseño de un computador en lo que se refiere a cómo satisfacer los niveles de prestaciones exigidos. El Capítulo 7 examina los mecanismos a través de los cuales un módulo de E/S interactúa con el resto del computador utilizando las técnicas de E/S programada, E/S por interrupciones, y acceso directo a memoria (DMA). También se describe la interfaz entre los módulos de E/S y los dispositivos externos.

CAPÍTULO 8. SISTEMAS OPERATIVOS

Un examen detallado de los sistemas operativos está fuera del ámbito de este libro. No obstante, es importante entender sus funciones básicas y cómo aprovecha el hardware para proporcionar el nivel de prestaciones deseado. El Capítulo 8 describe los principios básicos de los sistemas operativos, y discute las características de diseño específicas del hardware del computador orientadas a dar soporte al sistema operativo. El capítulo comienza con una breve introducción histórica, que sirve para identificar los principales tipos de sistemas operativos y presentar las razones por las que se utilizan. A continuación se explica la multiprogramación analizando las funciones de planificación (*scheduling*) a corto y largo plazo. Para terminar, se estudia la gestión de memoria incluyendo una discusión de los conceptos de segmentación, paginación, y memoria virtual.

CAPÍTULO 3

Perspectiva de alto nivel del funcionamiento y de las interconexiones del computador

PUNTOS CLAVE

- Un ciclo de instrucción consiste en la captación de la instrucción, seguida de ninguno o varios accesos a operandos, ninguno o varios almacenamientos de operandos, y la comprobación de las interrupciones (si estas están habilitadas).
- Los principales componentes del computador (procesador, memoria principal, módulos de E/S) necesitan estar interconectados para intercambiar datos y señales de control. El medio de interconexión más popular es un bus compartido constituido por un conjunto de líneas. En los computadores actuales, es usual utilizar una jerarquía de buses para mejorar el nivel de prestaciones.
- Los aspectos clave del diseño de los buses son el arbitraje (si el permiso para enviar las señales a través de las líneas del bus se controla de forma centralizada o distribuida); la temporización (sin las señales del bus se sincronizan mediante un reloj central o se envían asíncronamente); y la anchura (número de líneas de dirección y datos).

A alto nivel, un computador está constituido por CPU (unidad central de procesamiento), memoria, y unidades de E/S, con uno o varios módulos de cada tipo. Estos componentes se interconectan de modo que se pueda llevar a cabo la función básica del computador, que es ejecutar programas. Así, a este nivel, se puede describir un computador (1) mediante el comportamiento de cada uno de sus componentes, es decir, mediante los datos y las señales de control que un componente intercambia con los otros, y (2) mediante la estructura de interconexión y los controles necesarios para gestionar el uso de dicha estructura.

Esta visión de alto nivel en términos de estructura y funcionamiento es importante debido a su capacidad explicativa de cara a la comprensión de la naturaleza del computador. Igualmente importante es su utilidad para entender los cada vez más complejos problemas de evaluación de prestaciones. Entender la estructura y el funcionamiento a alto nivel permite hacerse una idea de los cuellos de botella del sistema, los caminos alternativos, la importancia de los fallos del sistema si hay un componente defectuoso, y la facilidad con que se pueden mejorar las prestaciones. En muchos casos, los requisitos de mayor potencia y capacidad de funcionamiento tolerante a fallos se satisfacen mediante cambios en el diseño más que con un incremento en la velocidad y en la fiabilidad de los componentes individuales.

Este capítulo se centra en las estructuras básicas utilizadas para la interconexión de los componentes del computador. A modo de revisión, el capítulo comienza con un somero examen de los componentes básicos y sus necesidades de interconexión. Después, se revisan los aspectos funcionales.

Más adelante estaremos preparados para analizar el uso de los buses que interconectan los componentes del sistema.

3.1. COMPONENTES DEL COMPUTADOR

Como se discutió en el Capítulo 2, virtualmente todos los computadores actuales se han diseñado basándose en los conceptos desarrollados por John von Neumann en el Instituto de Estudios

Avanzados (*Institute for Advances Studies*) de Priceton. Tal diseño se conoce con el nombre de *Arquitectura de von Neumann* y se basa en tres conceptos clave:

- Los datos y las instrucciones se almacenan en una sola memoria de lectura-escritura.
- Los contenidos de esta memoria se direccionan indicando su posición, sin considerar el tipo de dato contenido en la misma.
- La ejecución se produce siguiendo una secuencia de instrucción tras instrucción (a no ser que dicha secuencia se modifique explícitamente).

Las razones que hay detrás de estos conceptos se discutieron en el Capítulo 2 pero merecen ser resumidas aquí. Hay un conjunto pequeño de componentes lógicos básicos que pueden combinarse de formas diferentes para almacenar datos binarios y realizar las operaciones aritméticas y lógicas con esos datos. Si se desea realizar un cálculo concreto, es posible utilizar una configuración de componentes lógicos diseñada específicamente para dicho cálculo. Se puede pensar en el proceso de conexión de los diversos componentes para obtener la configuración deseada como si se tratase de una forma de programación. El «programa» resultante es hardware y se denomina *programa cableado (hardwired program).*

Considérese ahora la siguiente alternativa. Se construye una configuración de uso general de funciones lógicas y aritméticas. Este hardware realizará funciones diferentes según las señales de control aplicadas. En el caso del hardware específico, el sistema acepta datos y produce resultados (Figura 3.1a). Con el hardware de uso general, el sistema acepta datos y señales de control y produce

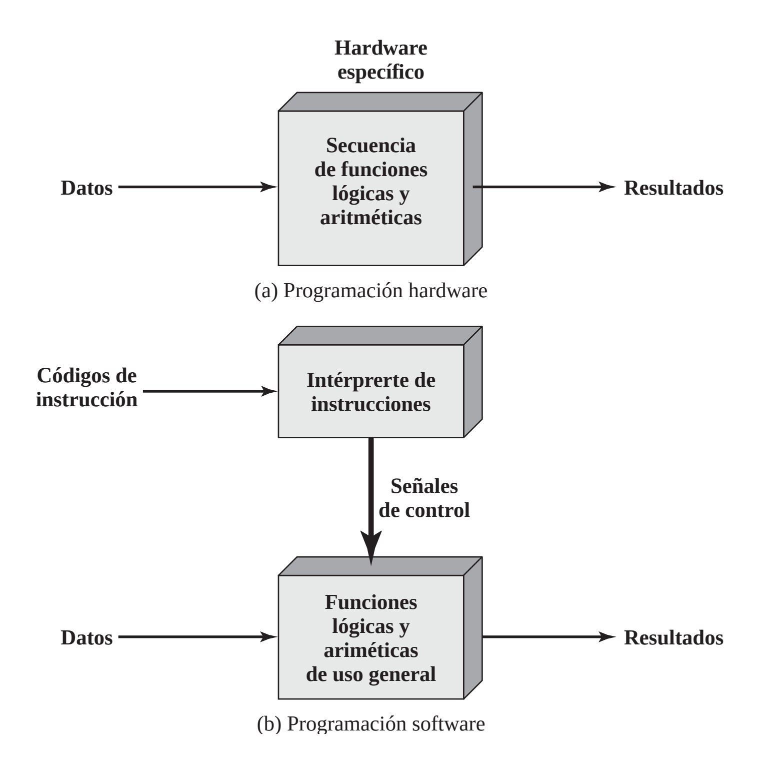


Figura 3.1. Alternativas hardware y software.

resultados. Así, en lugar de reconfigurar el hardware para cada nuevo programa, el programador simplemente necesita proporcionar un nuevo conjunto de señales de control.

¿Cómo se suministran las señales de control? La respuesta es simple pero ingeniosa. El programa es realmente una secuencia de pasos. En cada paso, se realiza una operación aritmética o lógica con ciertos datos. Para cada paso, se necesita un nuevo conjunto de señales de control. La solución consiste en asociar un código específico a cada posible conjunto de señales de control, y añadir al hardware de uso general una parte encargada de generar las señales de control a partir del código (Figura 3.1b).

Programar es ahora mucho más fácil. En lugar de tener que reconfigurar el hardware para cada programa, todo lo que se necesita es proporcionar una nueva secuencia de códigos. Cada código es, de hecho, una instrucción, y una parte del hardware interpreta cada instrucción y genera las señales de control. Para distinguir este nuevo método de programación, una secuencia de códigos o instrucciones se denomina *software*.

La Figura 3.1b muestra dos componentes esenciales del sistema: un intérprete de instrucciones y un módulo de uso general para las funciones aritméticas y lógicas. Estos dos elementos constituyen la CPU. Se requieren varios componentes adicionales para que el computador pueda funcionar. Los datos y las instrucciones deben introducirse en el sistema. Para eso se necesita algún tipo de módulo de entrada. Este módulo contiene los componentes básicos para captar datos e instrucciones en cierto formato y traducirlos al formato de señales que utiliza el sistema. Se necesita un medio para proporcionar los resultados, el módulo de salida. Globalmente, estos módulos se conocen con el nombre de *componentes de E/S (Entrada/Salida)*.

Se necesita un componente más. Un dispositivo de entrada proporcionará los datos y las instrucciones secuencialmente, uno tras otro. Pero un programa no siempre ejecuta las instrucciones según la misma secuencia; puede saltarse ciertas instrucciones (por ejemplo, al ejecutar la instrucción de salto IAS). De la misma forma, las operaciones con datos pueden necesitar acceder a más de un operando y según una secuencia determinada. Por ello, debe existir un sitio para almacenar temporalmente tanto las instrucciones como los datos. Ese módulo se llama *memoria*, o *memoria principal* para distinguirlo de los periféricos y la memoria externa. Von Neumann indicó que la misma memoria podría ser usada tanto para las instrucciones como para los datos.

La Figura 3.2 muestra estos componentes de alto nivel y sugiere las interacciones entre ellos. Típicamente, la CPU se encarga del control. Intercambia datos con la memoria. Para ello, usualmente utiliza dos registros internos (en la CPU): un registro de direcciones de memoria (MAR, *Memory Address Register*), que especifica la dirección en memoria de la próxima lectura o escritura, y un registro para datos de memoria (MBR, *Memory Buffer Register*), que contiene el dato que se va a escribir en memoria o donde se escribe el dato que se va a leer de memoria. Igualmente, un registro de direcciones de E/S (E/SAR, *E/S Address Register*) especifica un dispositivo de E/S. Un registro para datos de E/S (E/S BR, *E/S Buffer Register*) se utiliza para intercambiar datos entre un módulo de E/S y la CPU.

Un módulo de memoria consta de un conjunto de posiciones, designadas por direcciones numeradas secuencialmente. Cada posición contiene un número binario que puede ser interpretado como una instrucción o como un dato. Un módulo de E/S transfiere datos desde los dispositivos externos a la CPU y a la memoria, y viceversa. Contiene los registros (buffers) internos para almacenar los datos temporalmente, hasta que puedan enviarse.

Tras esta breve descripción de los principales componentes, revisaremos cómo funcionan estos cuando ejecutan programas.

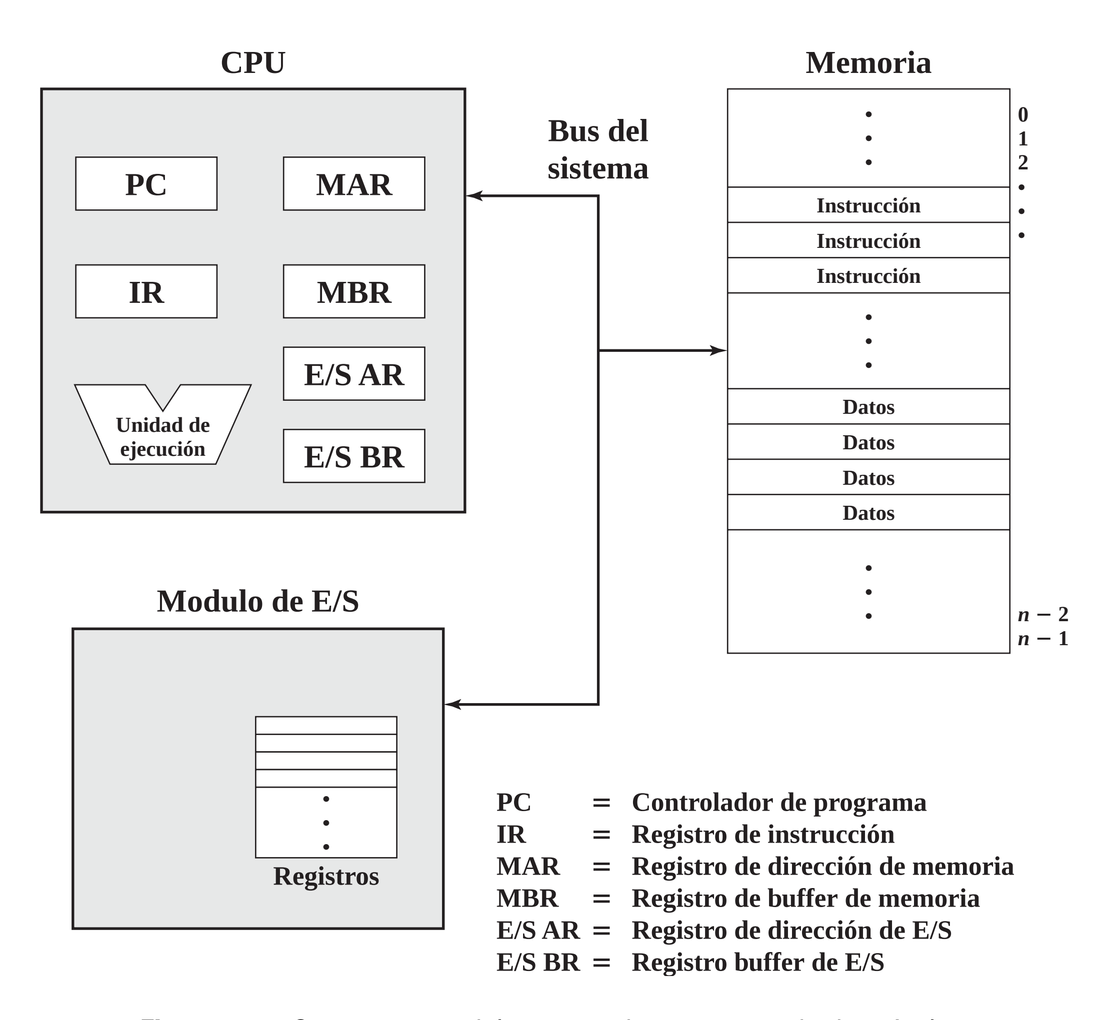


Figura 3.2. Componentes del computador: esquema de dos niveles.

3.2. FUNCIONAMIENTO DEL COMPUTADOR

La función básica que realiza un computador es la ejecución de un programa, constituido por un conjunto de instrucciones almacenadas en memoria. El procesador es precisamente el que se encarga de ejecutar las instrucciones especificadas en el programa. Esta sección proporciona una revisión de los aspectos clave en la ejecución de un programa, que en su forma más simple consta de dos etapas: El procesador lee (*capta*) la instrucción de memoria, y la ejecuta. La ejecución del programa consiste en la repetición del proceso de captación de instrucción y ejecución de instrucción. Por supuesto, la ejecución de la instrucción puede a su vez estar compuesta de cierto número de pasos (obsérvese, por ejemplo, la parte inferior de la Figura 2.4).

El procesamiento que requiere una instrucción se denomina *ciclo de instrucción*. Se representa en la Figura 3.3 utilizando la descripción simplificada de dos etapas explicada más arriba. Los dos pasos se denotan como *ciclo de captación* y *ciclo de ejecución*. La ejecución del programa se para solo si

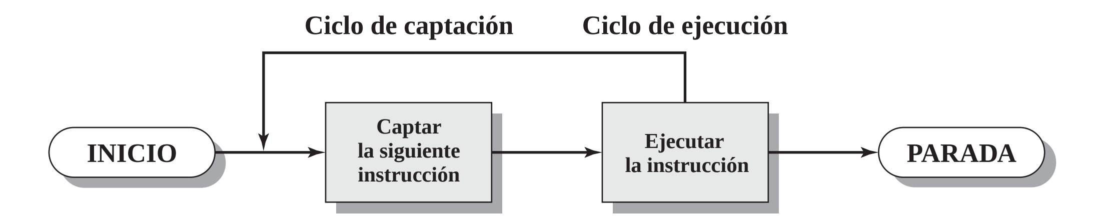


Figura 3.3. Ciclo de instrucción básico.

la máquina se desconecta, se produce algún tipo de error *irrecuperable* o ejecuta una instrucción del programa que detiene al computador.

LOS CICLOS DE CAPTACIÓN Y EJECUCIÓN

Al comienzo de cada ciclo de instrucción, la CPU capta una instrucción de memoria. En una CPU típica, se utiliza un registro llamado contador de programa (PC, *Program Counter*) para seguir la pista de la instrucción que debe captase a continuación. A no ser que se indique otra cosa, la CPU siempre incrementa el PC después de captar cada instrucción, de forma que captará la siguiente instrucción de la secuencia (es decir, la instrucción situada en la siguiente dirección de memoria). Considérese, por ejemplo, un computador en el que cada instrucción ocupa una palabra de memoria de 16 bits. Se supone que el contador de programa almacena el valor 300. La CPU captará la próxima instrucción almacenada en la posición 300. En los siguientes ciclos de instrucción, captará las instrucciones almacenadas en las posiciones 301, 302, 303, y así sucesivamente. Esta secuencia se puede alterar, como se explicará en breve.

La instrucción captada se almacena en un registro de la CPU conocido como registro de instrucción (IR, *Instruction Register*). La instrucción se escribe utilizando un código binario que especifica la acción que debe realizar la CPU. La CPU interpreta la instrucción y lleva a cabo la acción requerida. En general, esta puede ser de cuatro tipos:

- **Procesador-Memoria:** deben transferirse datos desde la CPU a la memoria, o desde la memoria a la CPU.
- **Procesador-E/S:** deben transferirse datos a o desde el exterior mediante transferencias entre la CPU y un módulo de E/S.
- **Procesamiento de Datos:** la CPU ha de realizar alguna operación aritmética o lógica con los datos.
- **Control:** una instrucción puede especificar que la secuencia de ejecución se altere (como la instrucción de salto IAS, Tabla 2.1). Por ejemplo, la CPU capta una instrucción de la posición 149 que especifica que la siguiente instrucción debe captarse de la posición 182. La CPU registrará este hecho poniendo en el contador de programa 182. Así, en el próximo ciclo de captación, la instrucción se cargará desde la posición 182 en lugar de desde la posición 150.

La ejecución de una instrucción puede implicar una combinación de estas acciones.

Considérese un ejemplo sencillo utilizando una máquina hipotética que incluye las caraterísticas enumeradas en la Figura 3.4. El procesador posee un único registro de datos llamado

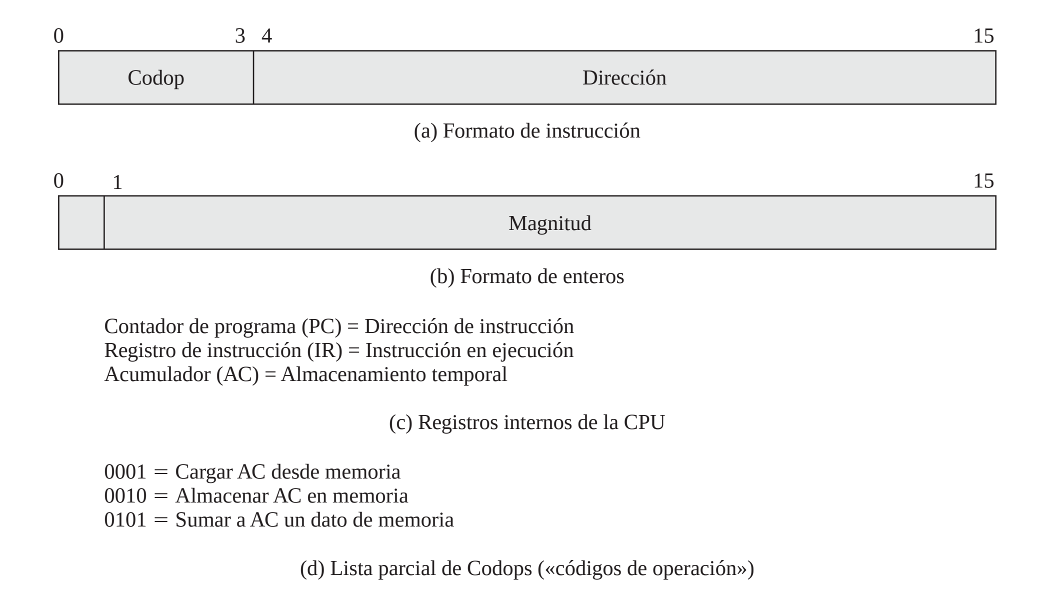


Figura 3.4. Características de una máquina hipotética.

acumulador (AC). Tanto las instrucciones como los datos son de 16 bits. Así, es conveniente organizar la memoria utilizando posiciones de 16 bits, o palabras. El formato de instrucción indica que puede haber $2^4 = 16$ códigos de operación (codops) diferentes, y se pueden direccionar directamente hasta $2^{12} = 4096$ (4K) palabras de memoria.

La Figura 3.5 ilustra la ejecución de una parte de un programa, mostrando las partes relevantes de la memoria y los registros de la CPU [1]. El fragmento de programa suma el contenido de la palabra de memoria en la dirección 940 con el contenido de la palabra de memoria en la dirección 941 y almacena el resultado en esta última posición. Se requieren tres instrucciones, que consumen tres ciclos de captación y tres de ejecución:

1. El contador de programa (PC) contiene el valor 300, la dirección de la primera instrucción. Esta instrucción (el valor hexadecimal 1940) se carga en el registro de instrucción (IR). Obsérvese que este proceso implicaría el uso del registro de dirección de memoria (MAR) y el registro de datos de memoria (MBR). Por simplicidad, se han ignorado estos registros intermedios.
2. Los primeros cuatro bits de IR (primer dígito hexadecimal) indican que el acumulador (AC) se va a cargar. Los restantes 12 bits (tres dígitos hexadecimales) especifican la dirección (940) que se va a cargar.
3. El registro PC se incrementa, y se capta la siguiente instrucción (5941) desde la dirección 301.
4. El contenido anterior de AC y el de la posición de memoria 941 se suman, y el resultado se almacena en AC.

[1] Se utiliza notación hexadecimal, en la que cada dígito representa cuatro bits. Esta es la notación más conveniente para representar los contenidos de la memoria y los registros cuando la longitud de palabra es múltiplo de 4 (por ejemplo, 8, 16, o 32). Para los lectores no familiarizados con este notación, se resume en el apéndice del Capítulo 8.

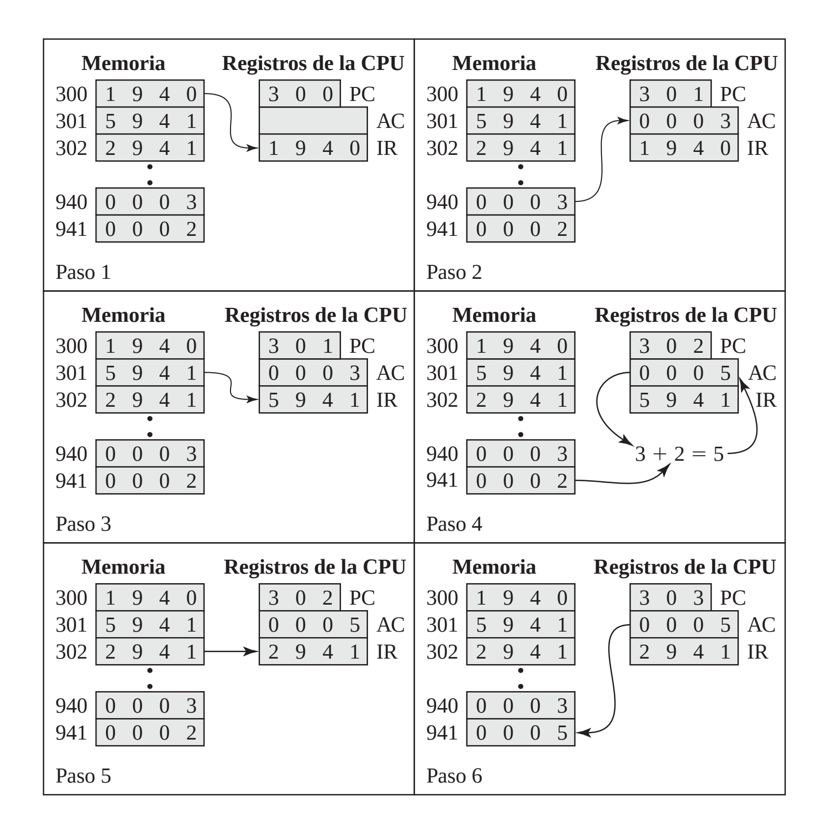


Figura 3.5. Ejemplo de ejecución de un programa (contenidos de la memoria y de los registros en hexadecimal).

5. El registro PC se incrementa, y se capta la siguiente instrucción (294) desde la posición 302.
6. El contenido de AC se almacena en la posición 941.

En este ejemplo, se necesitan tres ciclos de instrucción, cada uno con un ciclo de captación y un ciclo de ejecución, para sumar el contenido de la posición 940 y el contenido de la 941. Con un conjunto de instrucciones más complejo, se hubieran necesitado menos ciclos. Así, en algunos procesadores más antiguos se incluían instrucciones con más de una dirección. De esta forma, el ciclo de ejecución de una instrucción generaría más de una referencia a memoria. Además, en lugar de referencias a memoria, una instrucción puede especificar una operación de E/S.

Por ejemplo, la instrucción del PDP-11 expresada simbólicamente como ADD B,A almacena la suma de los contenidos de las posiciones B y A en la posición de memoria A. Se produce un solo ciclo de instrucción con los siguientes pasos:

- Se capta la instrucción ADD.
- El contenido de la posición de memoria A se lee y pasa al procesador
- El contenido de la posición de memoria B se lee y pasa al procesador. Para que el contenido de A no se pierda, el procesador debe tener al menos dos registros para almacenar valores de memoria, en lugar de un solo acumulador.

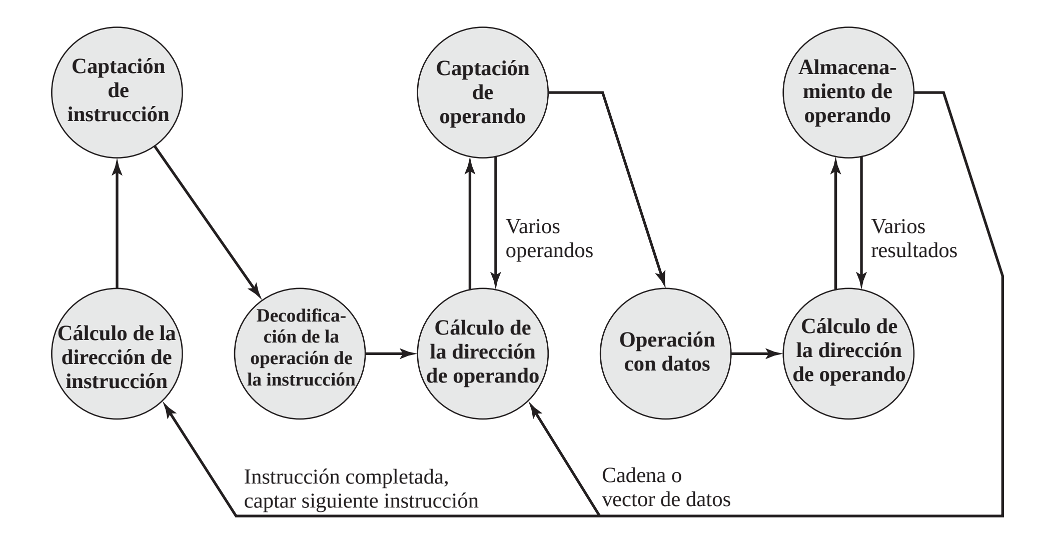


Figura 3.6. Diagrama de estados del ciclo de intrucción.

- Se suman los dos valores.
- El procesador escribe el resultado en la posición de memoria A.

Así, el ciclo de ejecución de una instrucción particular puede ocasionar más de una referencia a memoria. Además, en lugar de referencias a memoria, una instrucción puede especificar una operación de E/S. Con estas consideraciones adicionales en mente, la Figura 3.6 proporciona una visión más detallada del ciclo de instrucción básico de la Figura 3.3. La figura tiene la forma de un diagrama de estados. Para un ciclo de instrucción dado, algunos estados pueden no darse y otros pueden visitarse más de una vez. Los estados se describen a continuación:

- **Cálculo de la dirección de la instrucción (IAC, *Instruction Address Calculation*):** determina la dirección de la siguiente instrucción a ejecutar. Normalmente, esto implica añadir un número fijo a la dirección de la instrucción previa. Por ejemplo, si las instrucciones tienen un tamaño de 16 bits y la memoria se organiza en palabras de 16 bits, se suma 1 a la dirección previa. En cambio, si la memoria se organiza en bytes (8 bits) direccionables individualmente, entonces hay que sumar 2 a la dirección previa.
- **Captación de instrucción (if, *Instruction Fetch*):** la CPU lee la instrucción desde su posición en memoria.
- **Decodificación de la operación indicada en la instrucción (IOD, *Instruction Operation Decoding*):** analiza la instrucción para determinar el tipo de operación a realizar y el (los) operando(s) a utilizar.
- **Cálculo de la dirección del operando (OAC, *Operand Address Calculation*):** si la instrucción implica una referencia a un operando en memoria o disponible mediante E/S, determina la dirección del operando.
- **Captación de operando (OF, *Operand Fetch*):** capta el operando desde memoria o se lee desde el dispositivo de E/S.

- **Operación con los datos (DO, *Data Operation*):** realiza la operación indicada en la instrucción.
- **Almacenamiento de operando (OS, *Operand Store*):** escribe el resultado en memoria o lo saca a través de un dispositivo de E/S.

Los estados en la parte superior de la Figura 3.6 ocasionan intercambios entre la CPU y la memoria o un módulo de E/S. Los estados en la parte inferior del diagrama solo ocasionan operaciones internas a la CPU. El estado oac aparece dos veces, puesto que una instrucción puede ocasionar una lectura, una escritura, o ambas cosas. No obstante, la acción realizada en ese estado es la misma en ambos casos, y por eso sólo se necesita un único identificador de estado.

Obsérvese además que en el diagrama se considera la posibilidad de múltiples operandos y múltiples resultados puesto que se necesitan en algunas instrucciones de ciertas máquinas. Por ejemplo, la instrucción ADD A,B del PDP-11 da lugar a la siguiente secuencia de estados: iac,if,iod,oac,of,oac, of,do,oac,os.

Por último, en algunas máquinas, con una única instrucción se puede especificar una operación a realizar con un vector (matriz unidimensional) de números o con una cadena (matriz unidimensional) de caracteres. Como indica la Figura 3.6, esto implicaría una repetición de estados de captación y/o almacenamiento de operando.

INTERRUPCIONES

Prácticamente todos los computadores disponen de un mecanismo mediante el que otros módulos (E/S, memoria) pueden interrumpir el procesamiento normal de la CPU. La Tabla 3.1 enumera las clases de interrupciones más comunes. La naturaleza específica de estas interrupciones se examina en este libro más tarde, especialmente en los Capítulos 7 y 12. Sin embargo, necesitamos introducir el concepto ahora para comprender más claramente la esencia del ciclo de instrucción y los efectos de las interrupciones en la estructura de interconexión. En este momento, el lector no necesita conocer

Tabla 3.1. Clases de interrupciones.

Programa	Generadas por alguna condición que se produce como resultado de la ejecución de una instrucción, tal como desbordamiento aritmético (*overflow*), división por cero, intento de ejecutar una instrucción máquina inexistente e intento de acceder fuera del espacio de memoria permitido para el usuario.
Temporización	Generadas por un temporizador interno al procesador. Esto permite al sistema operativo realizar ciertas funciones de manera regular.
E/S	Generadas por un controlador de E/S, para indicar la finalización sin problemas de una operación o para avisar de ciertas condiciones de error.
Fallo de hardware	Generadas por un fallo tal como la falta de potencia de alimentación o un error de paridad en la memoria.

los detalles de la generación y el procesamiento de las interrupciones, sino solamente concentrarse en la comunicación entre módulos que resultan de las interrupciones.

En primer lugar las interrupciones proporcionan una forma de mejorar la eficiencia del procesador. Por ejemplo, la mayoría de los dispositivos externos son mucho más lentos que el procesador. Supóngase que el procesador está transfiriendo datos a una impresora utilizando el esquema del ciclo de instrucción de la Figura 3.3. Después de cada operación de escritura, el procesador tendrá que parar y permanecer ocioso hasta que la impresora complete la escritura. La longitud de esta pausa puede ser del orden de muchos cientos o incluso miles de ciclos de instrucción que no implican acceso a memoria. Claramente, esto supone un derroche en el uso del procesador.

La Figura 3.7a ilustra la situación del ejemplo referido en el párrafo precedente. El programa de usuario realiza una serie de llamadas de escritura (WRITE) entremezcladas con el procesamiento. Los segmentos de código 1, 2, y 3 corresponden a secuencias de instrucciones que no ocasionan operaciones de E/S. Las llamadas de escritura (WRITE) correponden a llamadas a un programa de E/S que es una de las utilidades del sistema operativo y que se encarga de la operación de E/S considerada. El programa de E/S está constituido por tres secciones:

- Una secuencia de instrucciones, rotulada con 4 en la figura, de preparación para la operación de E/S a realizar. Esto puede implicar la copia del dato que se va a proporcionar en un registro intermedio (buffer) especial, y preparar los parámetros de control del dispositivo de E/S.
- La orden de E/S propiamente dicha. Si no se utilizan interrupciones, una vez que se ejecuta esta orden, el programa debe esperar a que el dispositivo de E/S complete la operación solicitada. El programa esperaría comprobando repetidamente una condición que indique si se ha realizado la operación de E/S.
- Una secuencia de instrucciones, rotulada con 5 en la figura, que terminan la operación de E/S. Estas pueden incluir la activación de un indicador (*flag*) que señale si la operación se ha completado correctamente o con errores.

Debido a que la operación de E/S puede necesitar un tiempo relativamente largo, el programa de E/S debe detenerse a esperar que concluya dicha operación; por consiguiente, el programa de usuario estará parado en las llamadas de escritura (WRITE) durante un periodo de tiempo considerable.

Las Interrupciones y el ciclo de instrucción. Con el uso de interrupciones, el procesador puede dedicarse a ejecutar otras instrucciones mientras una operación de E/S está en curso. Considérese el flujo de control de la Figura 3.7b. Como antes, el programa de usuario llega a un punto en el que realiza una llamada al sistema para realizar una escritura (WRITE). El programa de E/S al que se llama en este caso está constituido solo por el código de preparación y la orden de E/S propiamente dicha. Después de que estas pocas instrucciones se hayan ejecutado, el control se devuelve al programa de usuario. Mientras tanto, el dispositivo externo está ocupado aceptando el dato de la memoria del computador e imprimiéndolo. Esta operación de E/S se realiza concurrentemente con la ejecución de instrucciones del programa de usuario.

Cuando el dispositivo externo pasa a estar preparado para actuar, es decir, cuando está listo para aceptar más datos del procesador, el módulo de E/S de este dispositivo externo envía una señal de *petición de interrupción* al procesador. El procesador responde suspendiendo la operación del programa que estaba ejecutando y salta a un programa, conocido como gestor de interrupción, que da

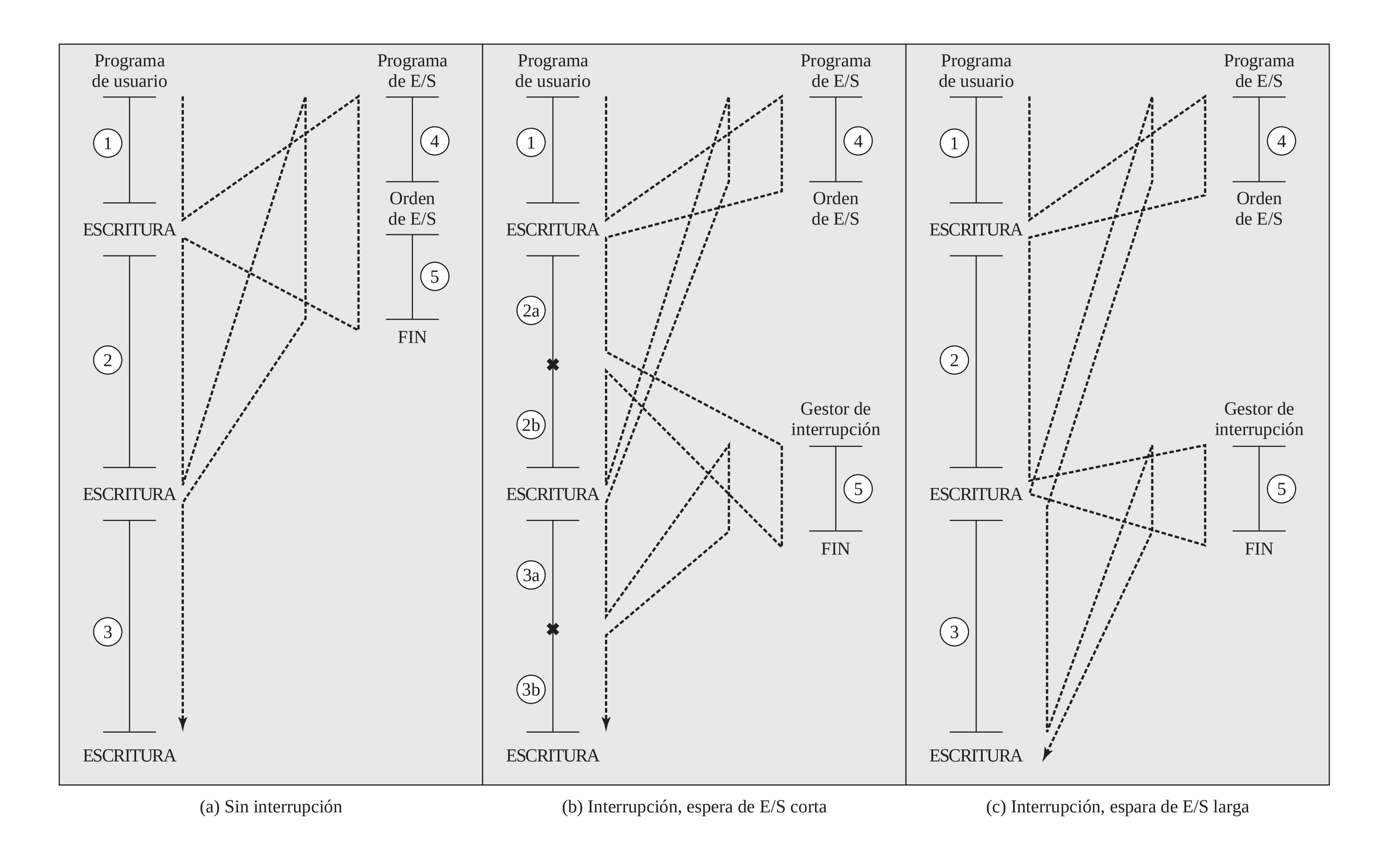


Figura 3.7. Flujo de control de un programa sin y con interrupción.

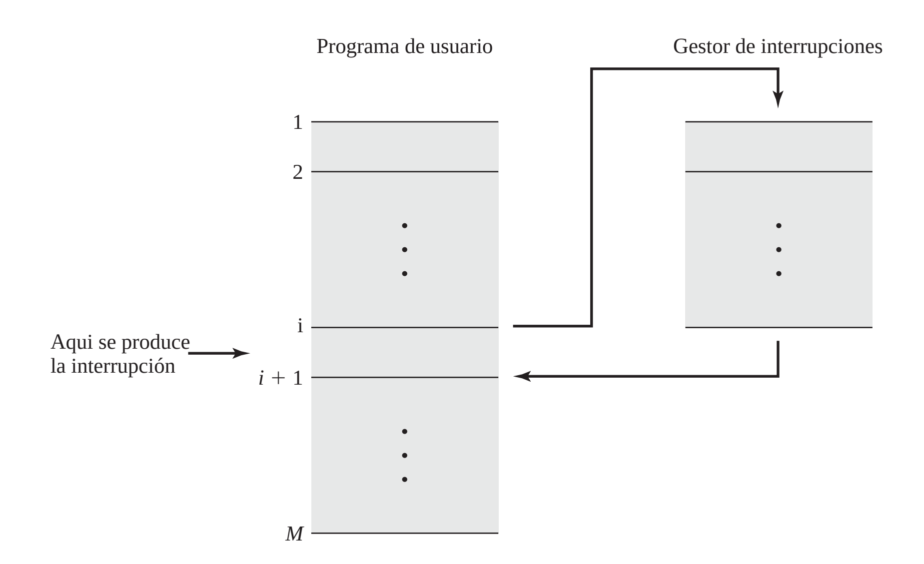


Figura 3.8. Transferencia de control debida a una interrupción.

servicio a ese dispositivo concreto, y prosigue con la ejecución del programa original después de haber dado dicho servicio al dispositivo. En la Figura 3.7b, los puntos en los que se producen las interrupciones se indican con una equis (X).

Desde el punto de vista del programa de usuario, una interrupción es precisamente eso: una interrupción en la secuencia normal de funcionamiento. Cuando el procesamiento de la interrupción se completa, la ejecución prosigue (Figura 3.8). Así, el programa de usuario no tiene que incluir ningún código especial para posibilitar las interrupciones; el procesador y el sistema operativo son los responsables de detener el programa de usuario y después permitir que prosiga en el mismo punto.

Para permitir el uso de interrupciones, se añade un *ciclo de interrupción* al ciclo de instrucción, como muestra la Figura 3.9. En el ciclo de interrupción, el procesador comprueba si se ha generado

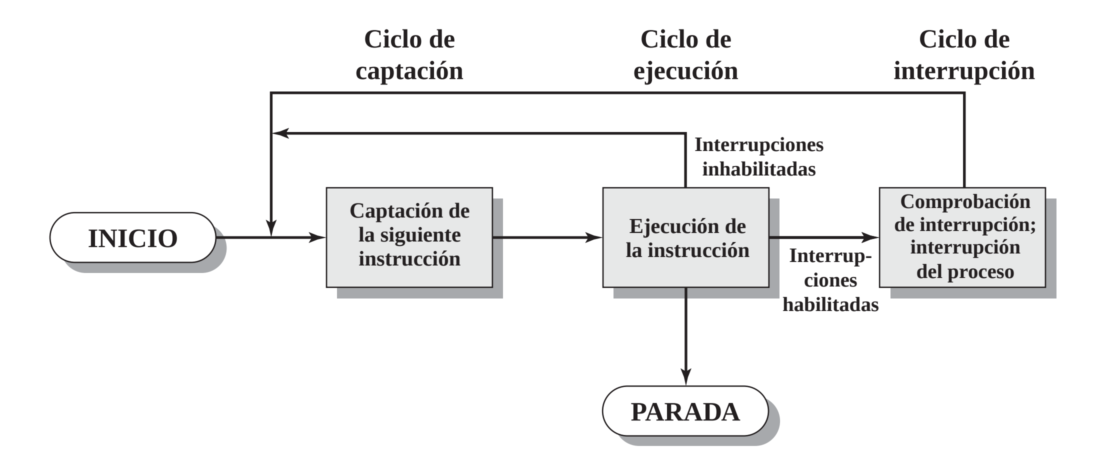


Figura 3.9. Ciclo de instrucción con interrupciones.

alguna interrupción, indicada por la presencia una señal de interrupción. Si no hay señales de interrupción pendientes, el procesador continúa con el ciclo de captación y accede a la siguiente instrucción del programa en curso. Si hay alguna interrupción pendiente, el procesador hace lo siguiente:

1. Suspende la ejecución del programa en curso y guarda su contexto. Esto significa almacenar la dirección de la siguiente instrucción a ejecutar (contenido actual del contador de programa) y cualquier otro dato relacionado con la actividad en curso del procesador.
2. Carga el contador de programa con la dirección de comienzo de una rutina de *gestión de interrupción.*

A continuación el procesador prosigue con el ciclo de captación y accede a la primera instrucción del programa de gestión de interrupción, que dará servicio a la interrupción. Generalmente el programa de gestión de interrupción forma parte del sistema operativo. Normalmente, este programa determina el origen de la interrupción y realiza todas las acciones que sean necesarias. Por ejemplo, en el caso que hemos estado analizando, el gestor determina qué módulo de E/S generó la interrupción, y puede saltar a un programa que escribe más datos en ese módulo de E/S. Cuando la rutina de gestión de interrupción se completa, el procesador puede proseguir la ejecución del programa de usuario en el punto en el que se interrumpió.

Es claro que este proceso supone una cierta penalización (*overhead*). Deben ejecutarse instrucciones extra (en el gestor de interrupción) para determinar el origen de la interrupción y para decidir la acción apropiada. No obstante, debido a la relativamente gran cantidad de tiempo que se perdería simplemente por la espera asociada a la operación de E/S, el procesador puede emplearse de manera mucho más eficiente utilizando interrupciones.

Para apreciar el aumento de eficiencia, considérese la Figura 3.10, que es un diagrama de tiempos basado en el flujo de control de las Figuras 3.7a y 3.7b. Las Figuras 3.7b y 3.10 asumen que el tiempo necesario para la operación de E/S es relativamente corto: menor que el tiempo para completar la ejecución de las instrucciones del programa de usuario que hay entre operaciones de escritura. La situación más frecuente, especialmente para un dispositivo lento como una impresora, es que la operación de E/S requiera mucho más tiempo para ejecutar una secuencia de instrucciones de usuario. La Figura 3.7c ilustra esta situación. En este caso, el programa de usuario llega a la segunda llamada de escritura (WRITE) antes de que la operación de E/S generada por la primera llamada se complete. El resultado es que el programa de usuario se detiene en este punto. Cuando la operación de E/S precedente se completa, esta nueva llamada de escritura se puede procesar, y se puede iniciar una nueva operación de E/S. La Figura 3.11 muestra la temporización para esta situación con y sin interrupciones. Podemos ver que existe una mejora de eficiencia porque parte del tiempo durante el cual la operación de E/S está en marcha se solapa con la ejecución de instrucciones de usuario.

La Figura 3.12 muestra el diagrama de estados del ciclo de instruccion modificado para incluir al procesamiento del ciclo de interrupción.

Interrupciones múltiples. Hasta ahora únicamente se ha discutido la existencia de una sola interrupción. Supóngase, no obstante, que se puedan producir varias interrupciones. Por ejemplo, un programa puede estar recibiendo datos a través de una línea de comunicación e imprimiendo resultados. La impresora generará interrupciones cada vez que complete una operación de escritura. El controlador de la línea de comunicación generará una interrupción cada vez que llegue una unidad de datos. La unidad de datos puede ser un carácter o un bloque, según el protocolo de comunicación. En

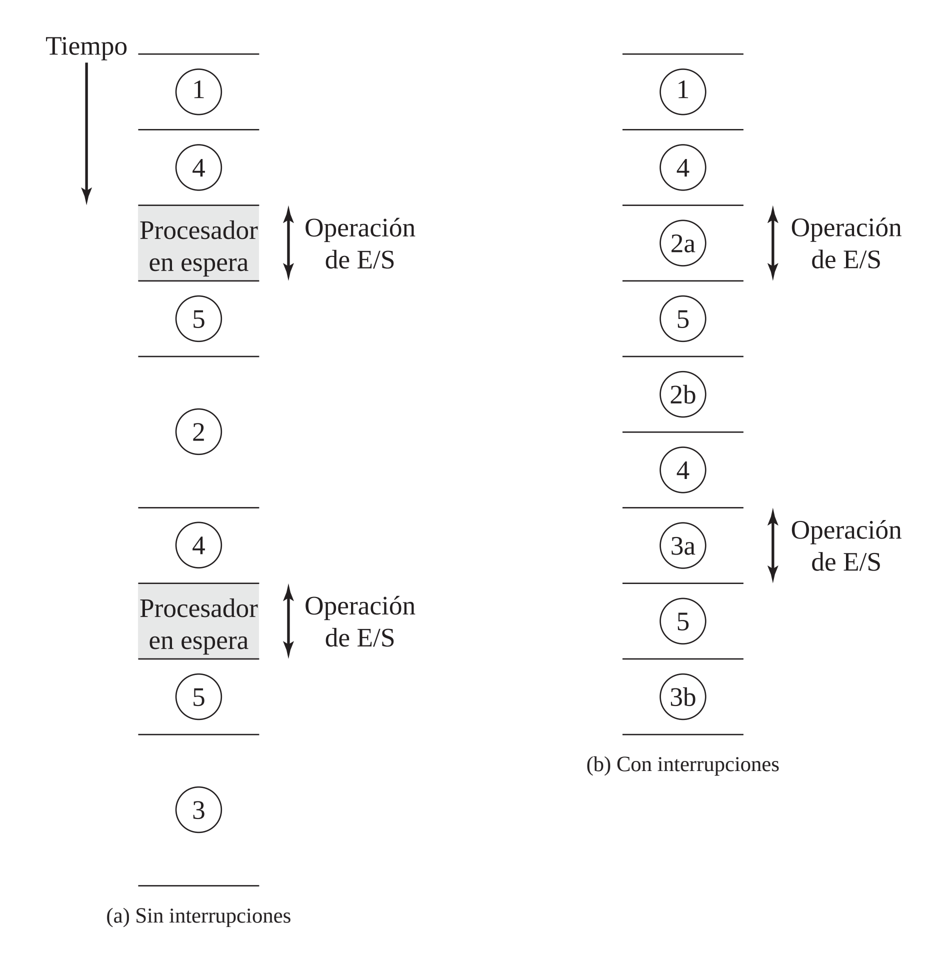


Figura 3.10. Temporización de un programa: espera corta de E/S.

cualquier caso, es posible que se produzca una interrupción de comunicaciones mientras se está procesando la interrupción de la impresora.

Se pueden seguir dos alternativas para tratar las interrupciones múltiples. La primera es desactivar las interrupciones mientras se está procesando una interrupción. Una *interrupción inhabilitada (disabled interrupt)* simplemente significa que el procesador puede y debe ignorar la señal de petición de interrupción. Si se produce una interrupción en ese momento, generalmente se mantiene pendiente y será examinada por el procesador una vez este haya activado las interrupciones. Así, cuando un programa de usuario se está ejecutando y se produce una interrupción, las interrupciones se inhabilitan inmediatamente. Después de que la rutina de gestión de interrupción termine, las interrupciones se habilitan antes de que el programa de usuario prosiga, y el procesador comprueba si se han producido interrupciones adicionales. Esta aproximación es correcta y simple, puesto que las interrupciones se manejan en un orden secuencial estricto (Figura 3.13a).

El inconveniente del enfoque anterior es que no tiene en cuenta la prioridad relativa ni las solicitudes con un tiempo crítico. Por ejemplo, cuando llega una entrada desde la línea de comunicaciones, esta debe tramitarse rápidamente para dejar espacio a los datos siguientes. Si los primeros datos no se han procesado antes de que lleguen los siguientes, se pueden perder.

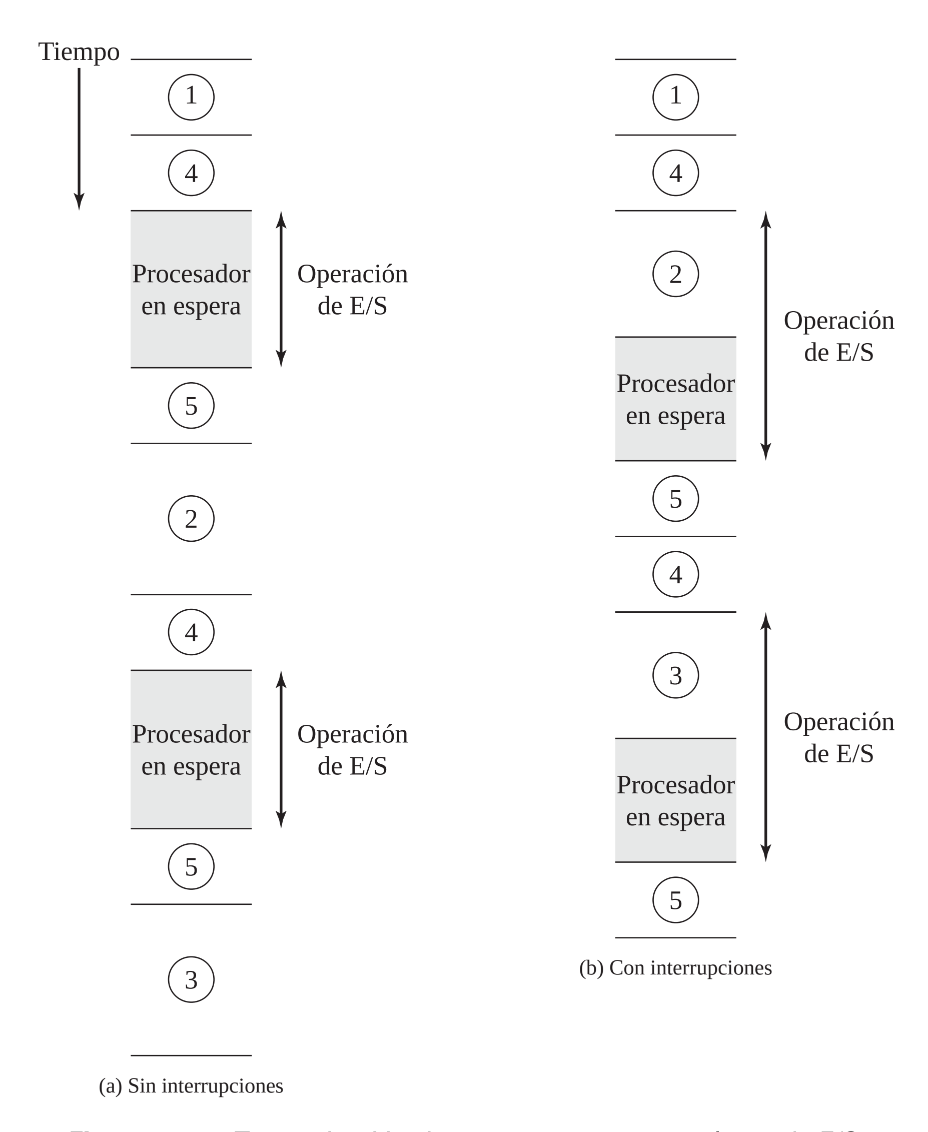


Figura 3.11. Temporización de un programa: espera larga de E/S.

Una segunda alternativa consiste en definir prioridades para las interrupciones y permitir que una interrupción de prioridad más alta pueda interrumpir a un gestor de interrupción de prioridad menor (Figura 3.13b). Como ejemplo de esta segunda alternativa, considérese un sistema con tres dispositivos de E/S: una impresora, un disco, y una línea de comunicaciones, con prioridades crecientes de 2, 4, y 5 respectivamente. La Figura 3.14, basada en un ejemplo de [TANE97] muestra una posible secuencia. Un programa de usuario comienza en $t = 0$. En $t = 10$, la impresora produce una interrupción; la información del programa de usuario se sitúa en la pila del sistema, y la ejecución continúa con la rutina de servicio de interrupción (ISR) de la impresora. Mientras se está ejecutando esta rutina, en $t = 15$, se produce una interrupción de comunicaciones. Como la línea de comunicaciones tiene una prioridad mayor que la impresora, se acepta la interrupción. La ISR de la impresora se interrumpe, su estado se introduce en la pila, y la ejecución continúa con la rutina de comunicaciones. Mientras se está ejecutando esta rutina, el disco ocasiona una interrupción ($t = 20$). Puesto que esta

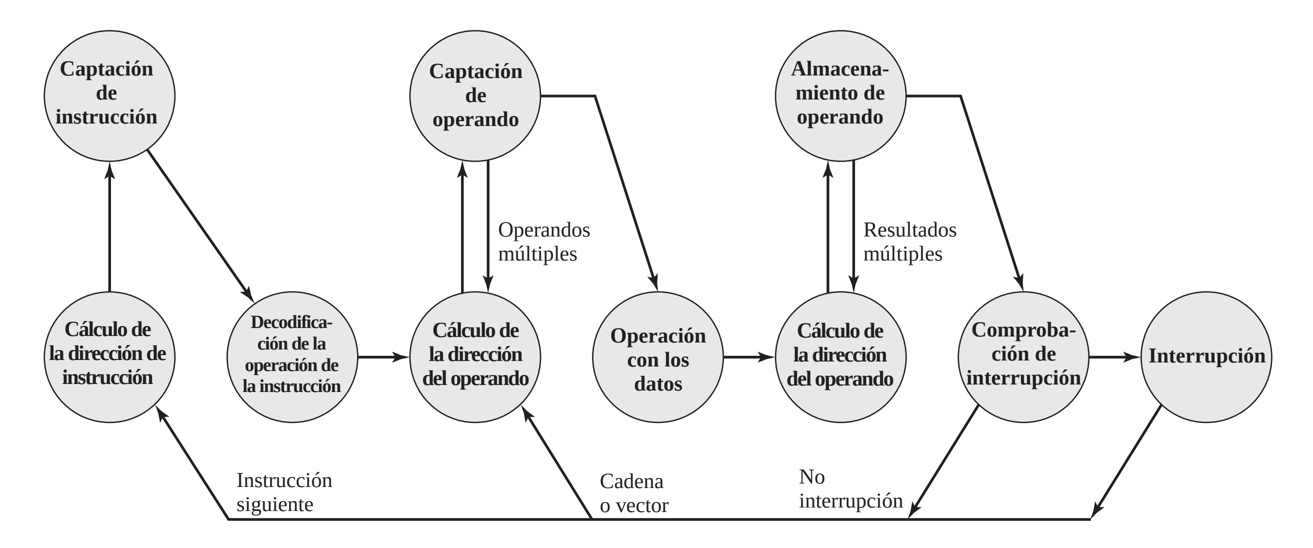


Figura 3.12. Diagrama de estados de un ciclo de instrucción, con interrupciones.

interrupción tiene una prioridad menor, simplemente se retiene, y la ISR de comunicaciones se ejecuta hasta que termina.

Cuando la rutina de comunicaciones termina ($t = 25$), se restaura el estado previo del procesador, que corresponde a la ejecución de la rutina de la impresora. No obstante, antes incluso de que pueda ejecutarse una sola instrucción de esa rutina, el procesador acepta la interrupción, de mayor prioridad, generada por el disco y el control se transfiere a la rutina de disco. Solo cuando esta rutina termina ($t = 35$), la rutina de la impresora puede reanudarse. Cuando termina esa rutina ($t = 40$), el control vuelve finalmente al programa de usuario.

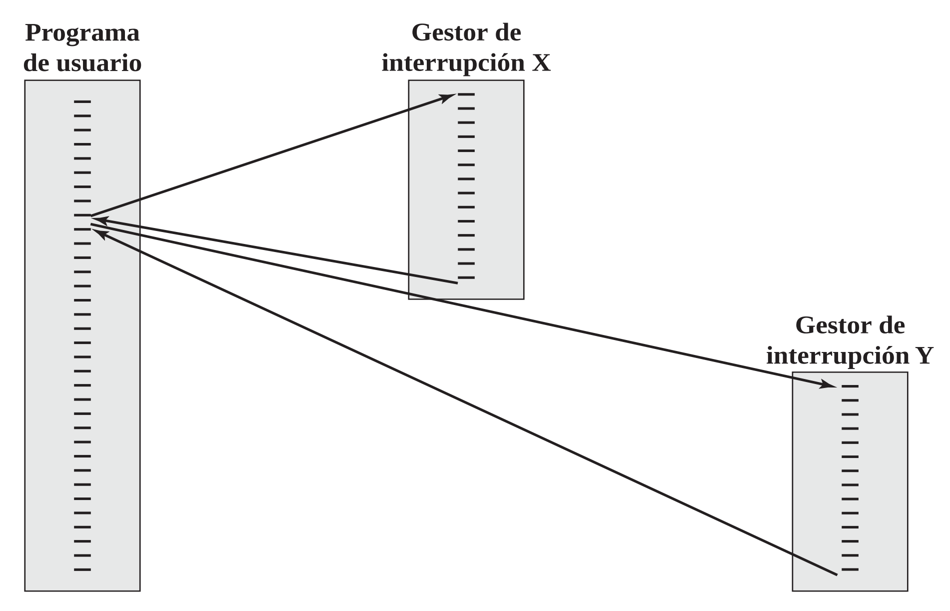
FUNCIONAMIENTO DE LAS E/S

Hasta aquí, hemos discutido el funcionamiento del computador controlado por el procesador, y nos hemos fijado esencialmente en la interacción del procesador y la memoria.

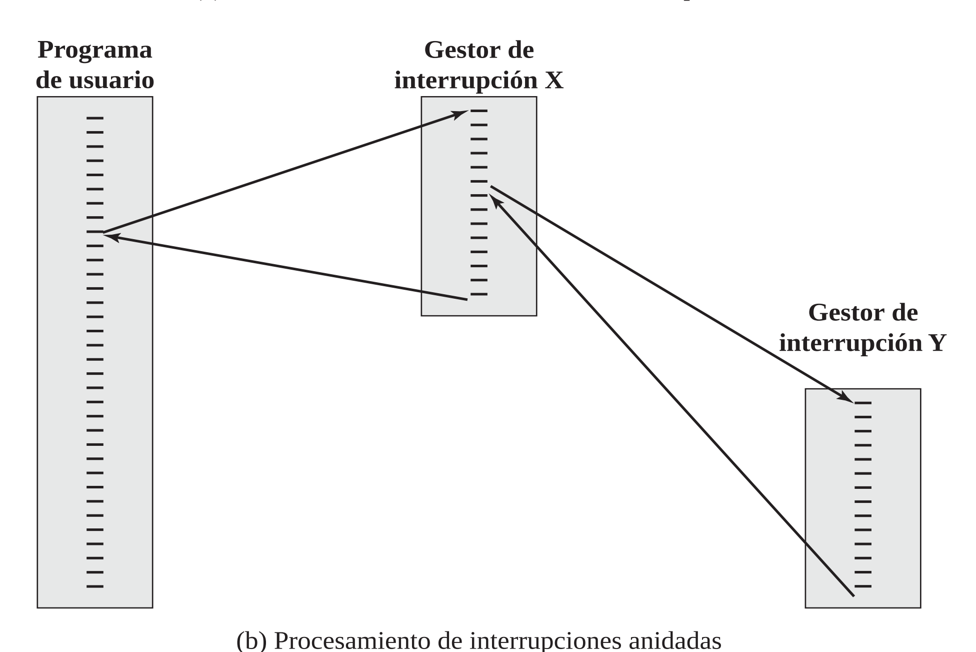
La discusión solo ha aludido al papel de los componentes de E/S. Este papel se discute con detalle en el Capítulo 7, pero es conveniente hacer aquí un breve resumen.

Un módulo de E/S (por ejemplo un controlador de disco) puede intercambiar datos directamente con el procesador. Igual que el procesador puede iniciar una lectura o escritura en memoria, especificando la dirección de una posición concreta de la misma, el procesador también puede leer o escribir datos de (o en) un módulo de E/S. En este último caso, el procesador identifica un dispositivo específico controlado por un módulo de E/S determinado. Por consiguiente, se puede producir una secuencia de instrucciones similar a la de la Figura 3.5, con instrucciones de E/S en lugar de las instrucciones de referencia a memoria.

En algunos casos, es deseable permitir que los intercambios de E/S se produzcan directamente con la memoria. En ese caso, el procesador cede a un módulo de E/S la autoridad para leer de o escribir en memoria, para que así la transferencia E/S-memoria pueda producirse sin la intervención del



(a) Procesamiento de una secuencia de interrupciones



(b) Procesamiento de interrupciones anidadas

Figura 3.13. Transferencia de control con varias interrupciones.

procesador. Durante esas transferencias, el módulo de E/S proporciona a la memoria las órdenes de lectura o escritura, liberando al procesador de cualquier responsabilidad en el intercambio. Esta operación se conoce con el nombre de acceso directo a memoria (DMA**,** *Direct Memory Access*), y se estudiará con detalle en el Capítulo 7.

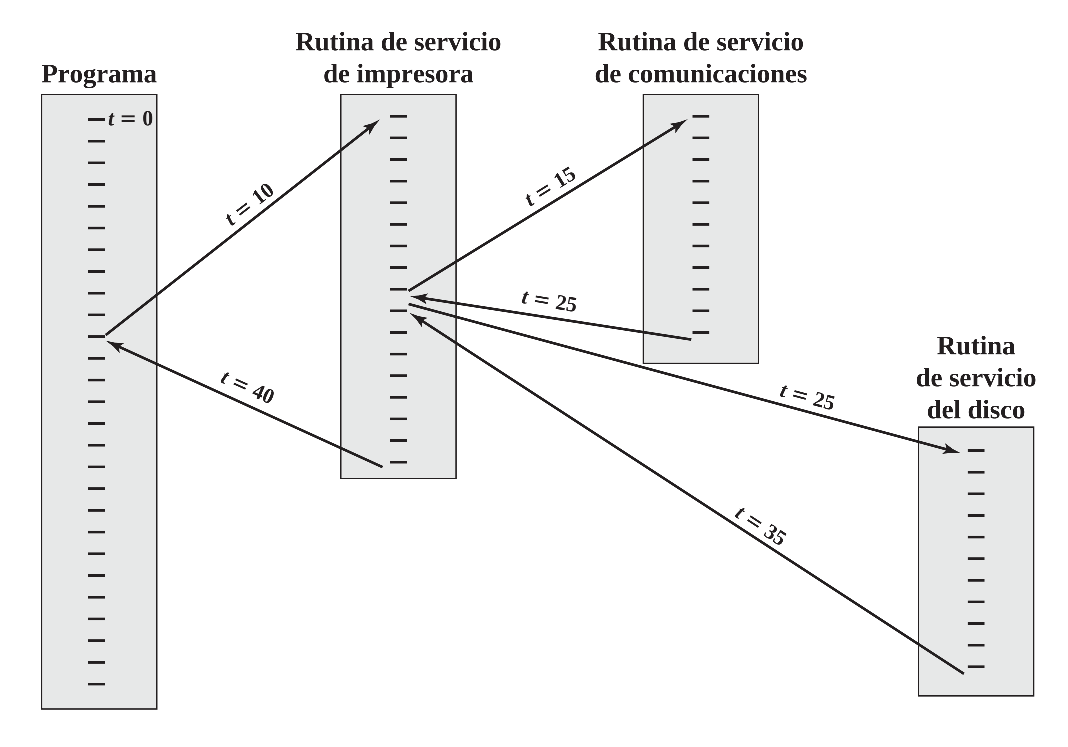


Figura 3.14. Ejemplo de secuencia temporal de varias interrupciones.

3.3. ESTRUCTURAS DE INTERCONEXIÓN

Un computador está constituido por un conjunto de unidades o módulos de tres tipos elementales (procesador, memoria, E/S) que se comunican entre sí. En efecto, un computador es una red de módulos elementales. Por consiguiente, deben existir lineas para interconectar estos módulos.

El conjunto de líneas que conectan los diversos módulos se denomina *estructura de interconexión*. El diseño de dicha estructura dependerá de los intercambios que deban producirse entre los módulos.

La Figura 3.15 sugiere los tipos de intercambios que se necesitan indicando las formas de las entradas y las salidas en cada tipo de módulo:

- **Memoria:** generalmente, un módulo de memoria está constituido por N palabras de la misma longitud. A cada palabra se le asigna una única dirección numérica $(0,1,..., N - 1)$. Una palabra de datos puede leerse de o escribirse en la memoria. El tipo de operación se indica mediante las señales de control *Read* (Leer) y *Write* (Escribir). La posición de memoria para la operación se especifica mediante una dirección.
- **Módulo de E/S:** desde un punto de vista interno (al computador), la E/S es funcionalmente similar a la memoria. Hay dos tipos de operaciones, leer y escribir. Además, un módulo de E/S puede controlar más de un dispositivo externo. Nos referiremos a cada una de estas interfaces con un dispositivo externo con el nombre de *puerto (port)*, y se le asignará una dirección a

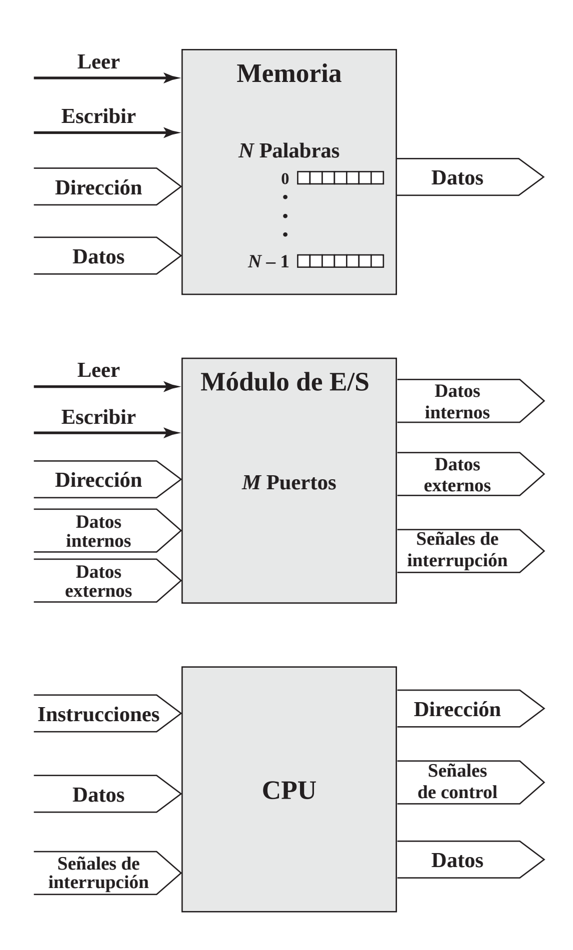


Figura 3.15. Módulos de un computador.

cada uno (0,1,...,M-1). Por otra parte, existen líneas externas de datos para la entrada y la salida de datos por un dispositivo externo. Por último, un módulo de E/S puede enviar señales de interrupción al procesador.

- **Procesador:** el procesador lee instrucciones y datos, escribe datos una vez los ha procesado, y utiliza ciertas señales para controlar el funcionamiento del sistema. También puede recibir señales de interrupción.

La lista precedente especifica los datos que se intercambian. La estructura de interconexión debe dar cobertura a los siguientes tipos de transferencias:

- **Memoria a procesador:** el procesador lee una instrucción o un dato desde la memoria.
- **Procesador a memoria:** el procesador escribe un dato en la memoria.
- **E/S a procesador:** el procesador lee datos de un dispositivo de E/S a través de un módulo de E/S.
- **Procesador a E/S:** el procesador envía datos al dispositivo de E/S.

- **Memoria a E/S y viceversa:** en estos dos casos, un módulo de E/S puede intercambiar datos directamente con la memoria, sin que tengan que pasar a través del procesador, utilizando el acceso directo a memoria (DMA).

A través de los años, se han probado diversas estructuras de interconexión. Las más comunes son, con diferencia, las estructuras de bus y de buses múltiples. El resto de este capítulo se dedica a evaluar las estructuras de buses.

3.4. INTERCONEXIÓN CON BUSES

Un bus es un camino de comunicación entre dos o más dispositivos. Una característica clave de un bus es que se trata de un medio de transmisión compartido. Al bus se conectan varios dispositivos, y cualquier señal transmitida por uno de esos dispositivos está disponible para que los otros dispositivos conectados al bus puedan acceder a ella. Si dos dispositivos transmiten durante el mismo periodo de tiempo, sus señales pueden solaparse y distorsionarse. Consiguientemente, solo un dispositivo puede transmitir con éxito en un momento dado.

Usualmente, un bus está constituido por varios caminos de comunicación, o líneas. Cada línea es capaz de transmitir señales binarias representadas por 1 y por 0. En un intervalo de tiempo, se puede transmitir una secuencia de dígitos binarios a través de una única línea. Se pueden utilizar varias líneas del bus para transmitir dígitos binarios simultáneamente (en paralelo). Por ejemplo, un dato de 8 bits puede transmitirse mediante ocho líneas del bus.

Los computadores poseen diferentes tipos de buses que proporcionan comunicación entre sus componentes a distintos niveles dentro de la jerarquía del sistema. El bus que conecta los componentes principales del computador (procesador, memoria, E/S) se denomina *bus del sistema (system bus)*. Las estructuras de interconexión más comunes dentro de un computador están basadas en el uso de uno o más buses del sistema.

ESTRUCTURA DEL BUS

El bus de sistema está constituido, usualmente, por entre cincuenta y cien líneas. A cada línea se le asigna un significado o una función particular. Aunque existen diseños de buses muy diversos, en todos ellos las líneas se pueden clasificar en tres grupos funcionales (Figura 3.16): líneas de datos, de

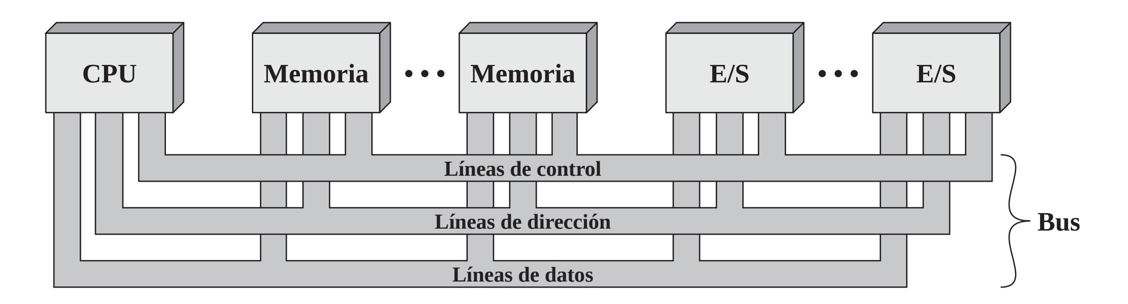


Figura 3.16. Esquema de interconexión mediante un bus.

direcciones y de control. Además, pueden existir líneas de alimentación para suministrar energía a los módulos conectados al bus.

Las *líneas de datos* proporcionan un camino para transmitir datos entre los módulos del sistema. El conjunto constituido por estas líneas se denomina *bus de datos*. El bus de datos puede incluir entre 32 y cientos de líneas, cuyo número se conoce como *anchura* del bus de datos. Puesto que cada línea solo puede transportar un bit cada vez, el número de líneas determina cuántos bits se pueden transferir al mismo tiempo. La anchura del bus es un factor clave a la hora de determinar las prestaciones del conjunto del sistema. Por ejemplo, si el bus de datos tiene una anchura de ocho bits, y las instrucciones son de 16 bits, entonces el procesador debe acceder al módulo de memoria dos veces por cada ciclo de instrucción.

Las *líneas de dirección* se utilizan para designar la fuente o el destino del dato situado en el bus de datos. Por ejemplo, si el procesador desea leer una palabra (8, 16 o 32 bits) de datos de la memoria, sitúa la dirección de la palabra deseada en las líneas de direcciones. Claramente, la anchura del bus de direcciones determina la máxima capacidad de memoria posible en el sistema. Además, las líneas de direcciones generalmente se utilizan también para direccionar los puertos de E/S. Usualmente, los bits de orden más alto se utilizan para seleccionar una posición de memoria o un puerto de E/S dentro de un módulo. Por ejemplo, en un bus de 8 bits, la dirección 01111111 e inferiores harían referencia a posiciones dentro de un módulo de memoria (el módulo 0) con 128 palabras de memoria, y las direcciones 10000000 y superiores designarían dispositivos conectados a un módulo de E/S (módulo 1).

Las **líneas de control** se utilizan para controlar el acceso y el uso de las líneas de datos y de direcciones. Puesto que las líneas de datos y de direcciones son compartidas por todos los componentes, debe existir una forma de controlar su uso. Las señales de control transmiten tanto órdenes como información de temporización entre los módulos del sistema. Las señales de temporización indican la validez de los datos y las direcciones. Las señales de órdenes especifican las operaciones a realizar. Algunas líneas de control típicas son

- **Escritura en memoria (*Memory write*):** hace que el dato del bus se escriba en la posición direccionada.
- **Lectura de memoria (*Memory read*):** hace que el dato de la posición direccionada se sitúe en el bus.
- **Escritura de E/S (*I/O write*):** hace que el dato del bus se transfiera a través del puerto de E/S direccionado.
- **Lectura de E/S (*E/S read*):** hace que el dato del puerto de E/S direccionado se sitúe en el bus.
- **Transferencia reconocida (*Transfer ACK*):** indica que el dato se ha aceptado o se ha situado en el bus.
- **Petición de bus (*Bus request*):** indica que un módulo necesita disponer del control del bus.
- **Cesión de bus (*Bus grant*):** indica que se cede el control del bus a un módulo que lo había solicitado.
- **Petición de interrupción (*Interrupt request*):** indica si hay una interrupción pendiente.
- **Interrupción reconocida (*Interrupt ACK*):** Señala que la interrupción pendiente se ha aceptado.

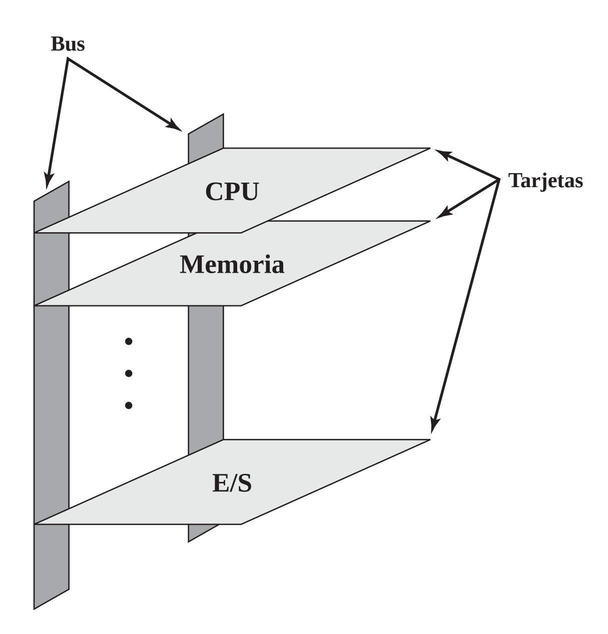


Figura 3.17. Implementación física típica de una arquitectura de bus.

- **Reloj (*clock*):** se utiliza para sincronizar las operaciones.
- **Inicio (*reset*):** pone los módulos conectados en su estado inicial.

El funcionamiento del bus se describe a continuación. Si un módulo desea enviar un dato a otro debe hacer dos cosas: (1) obtener el uso del bus y (2) transferir el dato a través del bus. Si un módulo desea pedir un dato a otro módulo, debe (1) obtener el uso del bus y (2) transferir la petición al otro módulo mediante las líneas de control y dirección apropiadas. Después debe esperar a que el segundo módulo envíe el dato.

Físicamente, el bus de sistema es de hecho un conjunto de conductores eléctricos paralelos. Estos conductores son líneas de metal grabadas en una tarjeta (tarjeta de circuito impreso). El bus se extiende a través de todos los componentes del sistema, cada uno de los cuales se conecta a algunas o a todas las líneas del bus. Una disposición física muy común se muestra en la Figura 3.17. En este ejemplo, el bus consta de dos columnas verticales de conductores. A lo largo de esas columnas, a intervalos regulares, hay puntos de conexión en forma de ranuras (*slots*) dispuestas en sentido horizontal para sostener las tarjetas de circuito impreso. Cada uno de los componentes principales del sistema ocupa una o varias tarjetas y se conecta al bus a través de esas ranuras. El sistema completo se introduce dentro de un chasis. Esta organización puede encontrarse todavía en alguno de los buses del computador. No obstante, los sistemas actuales tienden a tener sus componentes principales en la misma tarjeta y los circuitos integrados incluyen más elementos. Así, el bus que conecta el procesador y la memoria caché se integra en el microprocesador junto con el procesador y la caché (*on-chip*), y el bus que conecta el procesador con la memoria y otros componentes se incluye en la tarjeta (*on-board*).

Esta es la disposición más conveniente. Así se puede adquirir un computador pequeño y expandirlo (ampliar memoria, módulos de E/S) más adelante añadiendo más tarjetas. Si un componente de una tarjeta falla, la tarjeta puede quitarse y sustituirse fácilmente.

JERARQUÍAS DE BUSES MÚLTIPLES

Si se conecta un gran número de dispositivos al bus, las prestaciones pueden disminuir. Hay dos causas principales:

1. En general, a más dispositivos conectados al bus, mayor es el retardo de propagación. Este retardo determina el tiempo que necesitan los dispositivos para coordinarse en el uso del bus. Si el control del bus pasa frecuentemente de un dispositivo a otro, los retardos de propagación pueden afectar sensiblemente a las prestaciones.
2. El bus puede convertirse en un cuello de botella a medida que las peticiones de transferencia acumuladas se aproximan a la capacidad del bus. Este problema se puede resolver en alguna medida incrementando la velocidad a la que el bus puede transferir los datos y utilizando buses más anchos (por ejemplo incrementando el bus de datos de 32 a 64 bits). Sin embargo, puesto que la velocidad de transferencia que necesitan los dispositivos conectados al bus (por ejemplo, controladores de gráficos y de vídeo, interfaces de red) está incrementándose rápidamente, es un hecho que el bus único está destinado a dejar de utilizarse.

Por consiguiente, la mayoría de los computadores utilizan varios buses, normalmente organizados jerárquicamente. Una estructura típica se muestra en la Figura 3.18a. Hay un bus local que conecta el procesador a una memoria caché y al que pueden conectarse también uno o más dispositivos locales. El controlador de memoria caché conecta la caché no solo al bus local sino también al bus de sistema, donde se conectan todos los módulos de memoria principal. Como se discute en el Capítulo 4, el uso de una caché alivia la exigencia de soportar los accesos frecuentes del procesador a memoria principal. De hecho, la memoria principal puede pasar del bus local al bus de sistema. De esta forma, las transferencias de E/S con la memoria principal a través del bus de sistema no interfieren la actividad del procesador.

Es posible conectar controladores de E/S directamente al bus de sistema. Una solución más eficiente consiste en utilizar uno o más buses de expansión. La interfaz del bus de expansión regula las transferencias de datos entre el bus de sistema y los controladores conectados al bus de expansión. Esta disposición permite conectar al sistema una amplia variedad de dispositivos de E/S y al mismo tiempo aislar el tráfico de información entre la memoria y el procesador del tráfico correspondiente a las E/S.

La Figura 3.18a muestra algunos ejemplos típicos de dispositivos de E/S que pueden estar conectados al bus de expansión. Las conexiones a red incluyen conexiones a redes de área local (LAN, *Local Area Networks*) tales como una red Ethernet de diez Mbps y conexiones a redes de area amplia (*Wide Area Networks*) tales como la red de conmutación de paquetes (*packet-switching network*). La interfaz SCSI (*Small Computer System Interface*) es en sí un tipo de bus utilizado para conectar controladores de disco y otros periféricos. El puerto serie puede utilizarse para conectar una impresora o un escáner.

Esta arquitectura de buses tradicional es razonablemente eficiente, pero muestra su debilidad a medida que los dispositivos de E/S ofrecen prestaciones cada vez mayores. La respuesta común a esta situación, por parte de la industria, ha sido proponer un bus de alta velocidad que está estrechamente integrado con el resto del sistema, y requiere solo un adaptador (*bridge*) entre el bus del procesador y el bus de alta velocidad. En algunas ocasiones, esta disposición es conocida como arquitectura de entreplanta (*mezzanine architecture*).

La Figura 3.18b muestra un ejemplo típico de esta aproximación. De nuevo, hay un bus local que conecta el procesador a un controlador de caché, que a su vez está conectado al bus de sistema

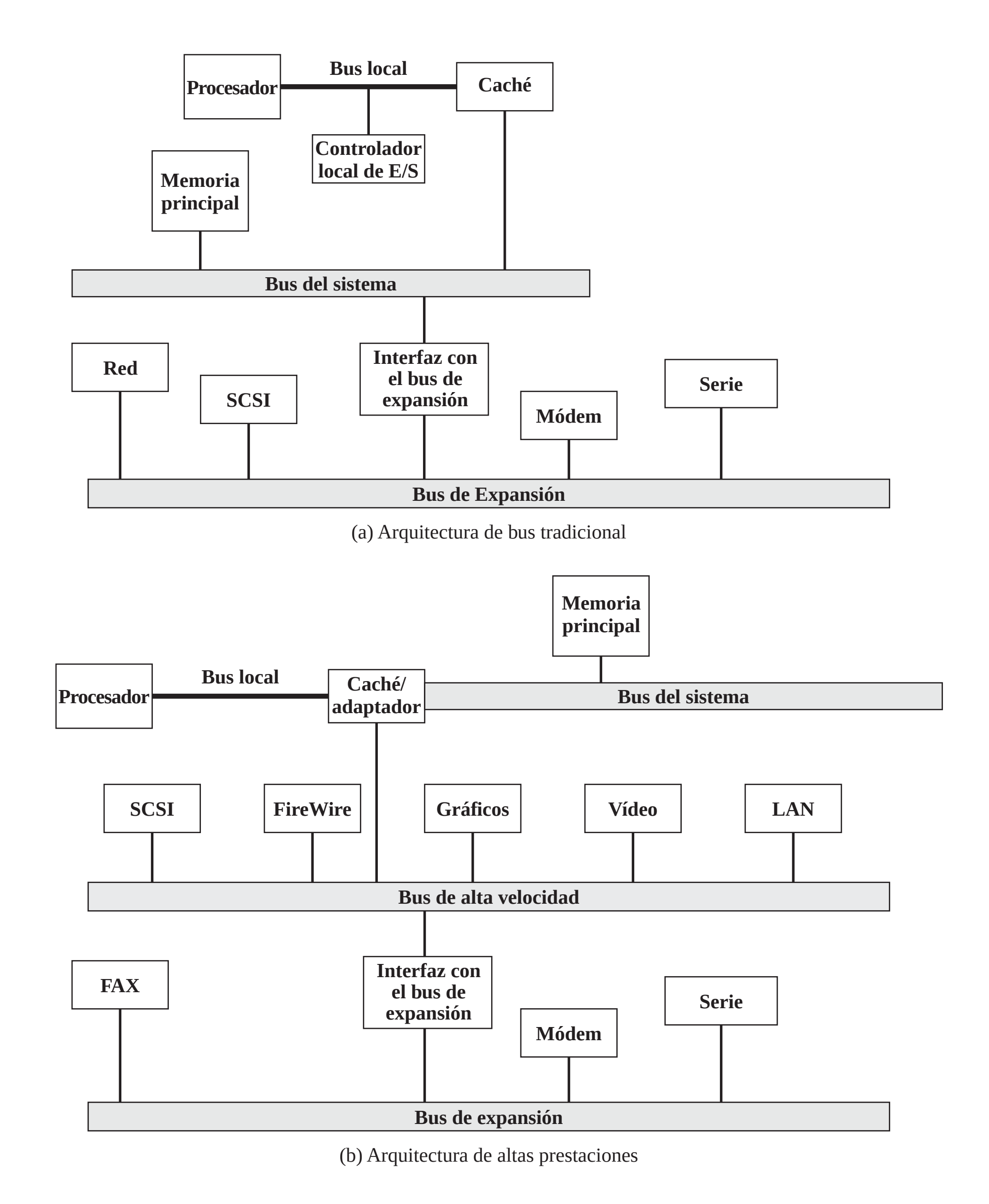


Figura 3.18. Ejemplos de configuraciones de bus.

que soporta a la memoria principal. El controlador de caché está integrado junto con el adaptador, o dispositivo de acoplo, que permite la conexión al bus de alta velocidad. Este bus permite la conexión de LAN de alta velocidad, tales como Fast Ethernet a cien Mbps, controladores de estaciones de trabajo específicos para aplicaciones gráficas y de vídeo, y también controladores de interfaz para buses de periféricos tales como SCSI y Firewire. Este último es un bus de alta velocidad diseñado específicamente para conectar dispositivos de E/S de alta capacidad. Los dispositivos de velocidad menor pueden conectarse al bus de expansión, que utiliza una interfaz para adaptar el tráfico entre el bus de expansión y el bus de alta velocidad.

La ventaja de esta organización es que el bus de alta velocidad acerca al procesador los dispositivos que exigen prestaciones elevadas y al mismo tiempo es independiente del procesador. Así, se pueden tolerar las diferencias de velocidad entre el procesador y el bus de altas prestaciones y las variaciones en la definición de las líneas de los buses. Los cambios en la arquitectura del procesador no afectan al bus de alta velocidad, y viceversa.

ELEMENTOS DE DISEÑO DE UN BUS

Aunque existe una gran diversidad de diseños de buses, hay unos pocos parámetros o elementos de diseño que sirven para distinguir y clasificar los buses. La Tabla 3.2 enumera los elementos clave.

Tabla 3.2. Elementos de diseño de un bus.

Tipo	**Anchura del bus**
Dedicado	Dirección
Multiplexado	Datos
Método de arbitraje	**Tipo de transferencia de datos**
Centralizado	Lectura
Distribuido	Escritura
Temporización	Lectura-modificación-escritura
Síncrono	Lectura-después de-escritura
Asíncrono	Bloque

Tipos de buses. Las líneas del bus se pueden dividir en dos tipos genéricos: dedicadas y multiplexadas. Una línea de bus dedicada está permanente asignada a una función o a un subconjunto físico de componentes del computador.

Un ejemplo de dedicación funcional, común en muchos buses, es el uso de líneas separadas para direcciones y para datos. Sin embargo, no es esencial. Por ejemplo, la información de dirección y datos podría transmitirse a través del mismo conjunto de líneas si se utiliza una línea de control de Dirección Válida. Al comienzo de la transferencia de datos, la dirección se sitúa en el bus y se activa la línea de Dirección Válida. En ese momento, cada módulo dispone de un periodo de tiempo para copiar la dirección y determinar si es él el módulo direccionado. Después la dirección se quita del bus, y las mismas conexiones se utilizan para la subsecuente tranferencia de lectura o escritura de datos. Este método de uso de las mismas líneas para usos diferentes se llama *multiplexado en el tiempo*.

La ventaja del multiplexado en el tiempo es el uso de menos líneas, cosa que ahorra espacio y, normalmente, costes. La desventaja es que se necesita una circuitería más compleja en cada módulo. Además, existe una posible reducción en las prestaciones debido a que los eventos que deben compartir las mismas líneas no pueden producirse en paralelo.

La *dedicación física* se refiere al uso de múltiples buses, cada uno de los cuales conecta solo un subconjunto de módulos. Un ejemplo típico es el uso de un bus de E/S para interconectar todos los módulos de E/S; este bus a su vez se conecta al bus principal a través de algún tipo de módulo adaptador de E/S. La ventaja potencial de la dedicación física es su elevado rendimiento, debido a que hay

menos conflictos por el acceso al bus (*bus contention*). Una desventaja es el incremento en el tamaño y el costo del sistema.

Método de arbitraje. En todos los sistemas, exceptuando los más simples, más de un módulo puede necesitar el control del bus. Por ejemplo, un módulo de E/S puede necesitar leer o escribir directamente en memoria, sin enviar el dato al procesador. Puesto que en un instante dado solo una unidad puede transmitir a través del bus, se requiere algún método de arbitraje. Los diversos métodos se pueden clasificar aproximadamente como centralizados o distribuidos. En un esquema centralizado, un único dispositivo hardware, denominado *controlador del bus* o *árbitro*, es responsable de asignar tiempos en el bus. El dispositivo puede estar en un módulo separado o ser parte del procesador. En un esquema distribuido, no existe un controlador central. En su lugar, cada módulo dispone de lógica para controlar el acceso y los módulos actúan conjuntamente para compartir el bus. En ambos métodos de arbitraje, el propósito es designar un dispositivo, el procesador o un módulo de E/S como maestro del bus. El maestro podría entonces iniciar una transferencia de datos (lectura o escritura) con otro dispositivo, que actúa como esclavo en este intercambio concreto.

Temporización. El término temporización hace referencia a la forma en la que se coordinan los eventos en el bus. Los buses utilizan temporización síncrona o asíncrona.

Con temporización síncrona, la presencia de un evento en el bus está determinada por un reloj. El bus incluye una línea de reloj a través de la que se transmite una secuencia en la que se alternan intervalos regulares de igual duración a uno y a cero. Un único intervalo a uno seguido de otro a cero se conoce como *ciclo de reloj* o *ciclo de bus* y define un intervalo de tiempo unidad (*time slot*). Todos los dispositivos del bus pueden leer la línea de reloj, y todos los eventos empiezan al principio del ciclo de reloj. La Figura 3.19 muestra el diagrama de tiempos de una operación de lectura síncrona (en el Apéndice 3A se puede consultar una descripción de los diagramas de tiempo). Otras señales del bus pueden cambiar en el flanco de subida de la señal de reloj (reaccionan con un ligero retardo). La mayoría de los eventos se prolongan durante un único ciclo de reloj. En este ejemplo sencillo, la CPU activa una señal de lectura y sitúa una dirección de memoria en las líneas de dirección durante el primer ciclo y puede activar varias líneas de estado. Una vez que las líneas se han estabilizado, el procesador activa una señal de inicio para indicar la presencia de la dirección en el bus. En el caso de una lectura, el procesador pone una orden de lectura al comienzo del segundo ciclo. El módulo de memoria reconoce la dirección y, después de un retardo de un ciclo, sitúa el dato en las líneas de datos. El procesador lee el dato de dichas líneas y quita la señal de lectura. En el caso de una escritura, el procesador pone el dato en las líneas de datos al comienzo del segundo ciclo, y pone la orden de escritura una vez estabilizadas las líneas de datos. El módulo de memoria copia la información de las líneas de datos durante el tercer ciclo de reloj.

Con la temporización asíncrona, la presencia de un evento en el bus es consecuencia y depende de que se produzca un evento previo. En el ejemplo sencillo de la Figura 3.20a, el procesador sitúa las señales de dirección y lectura en el bus. Después de un breve intervalo para que las señales se estabilicen, activa la orden de lectura, indicando la presencia de señales de dirección y control válidas. El módulo de memoria correspondiente decodifica la dirección y responde proporcionando el dato en la línea de datos. Una vez estabilizadas las líneas de datos, el módulo de memoria activa la línea de *reconocimiento* para indicar al procesador que el dato está disponible. Cuando el maestro ha leído el dato de las líneas correspondientes, deshabilita la señal de lectura. Esto hace que el módulo de memoria libere las líneas de datos y reconocimiento. Por último, una vez se ha desactivado la línea de reconocimiento, el procesador quita la información de dirección de las líneas correspondientes.

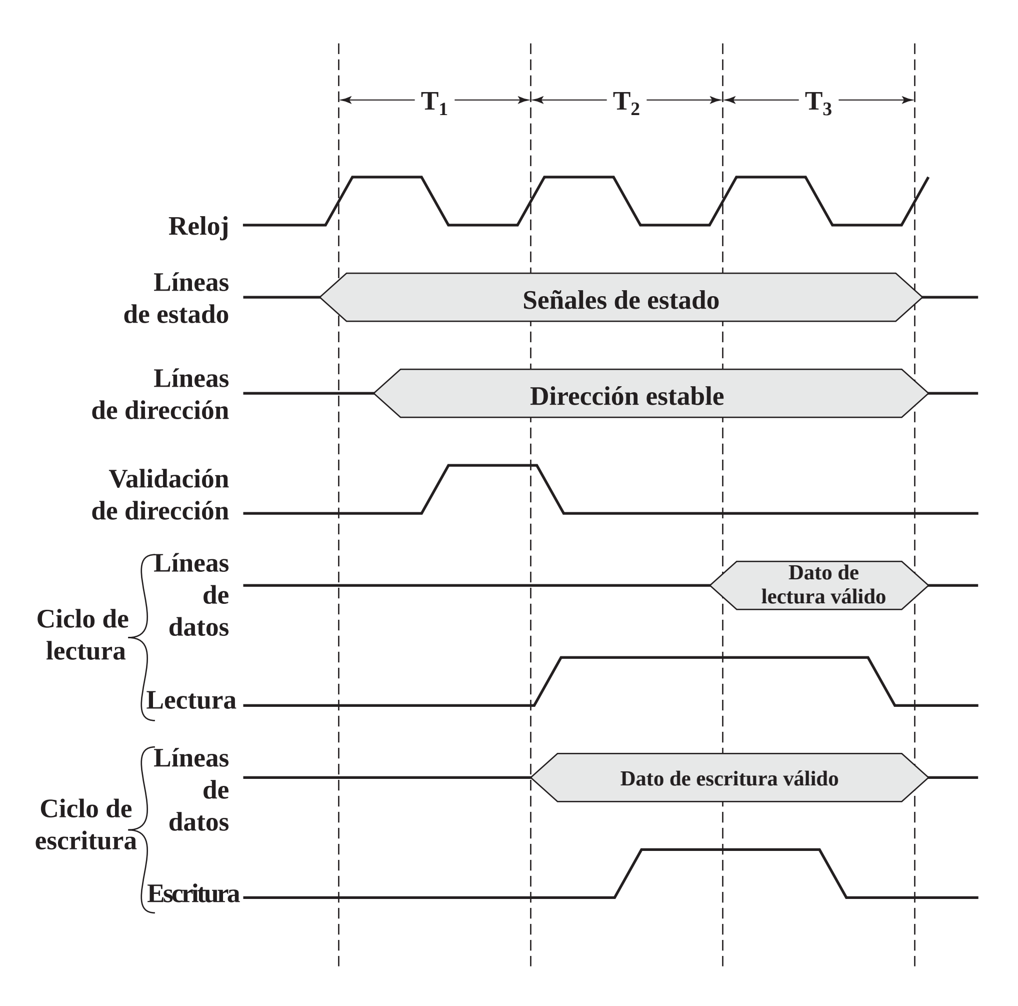


Figura 3.19. Temporización síncrona de las operaciones de bus.

La Figura 3.20b muestra una operación sencilla de escritura asíncrona. En este caso, el maestro sitúa el dato en las líneas de datos al mismo tiempo que la dirección y la información de estado en las líneas correspondientes. El módulo de memoria responde a la orden de escritura copiando el dato de las líneas de datos y activando la línea de reconocimiento. Entonces, el maestro retira la señal de escritura y el módulo de memoria la señal de reconocimiento.

La temporización síncrona es más fácil de implementar y comprobar. Sin embargo, es menos flexible que la temporización asíncrona. Debido a que todos los dispositivos en un bus síncrono deben utilizar la misma frecuencia de reloj, el sistema no puede aprovechar las mejoras en las prestaciones de los dispositivos. Con la temporización asíncrona, pueden compartir el bus una mezcla de dispositivos lentos y rápidos, utilizando tanto las tecnologías más antiguas, como las más recientes.

Anchura del bus. El concepto de anchura del bus se ha presentado ya. La anchura del bus de datos afecta a las prestaciones del sistema: cuanto más ancho es el bus de datos, mayor es el número de bits que se transmiten a la vez. La anchura del bus de direcciones afecta a la capacidad del

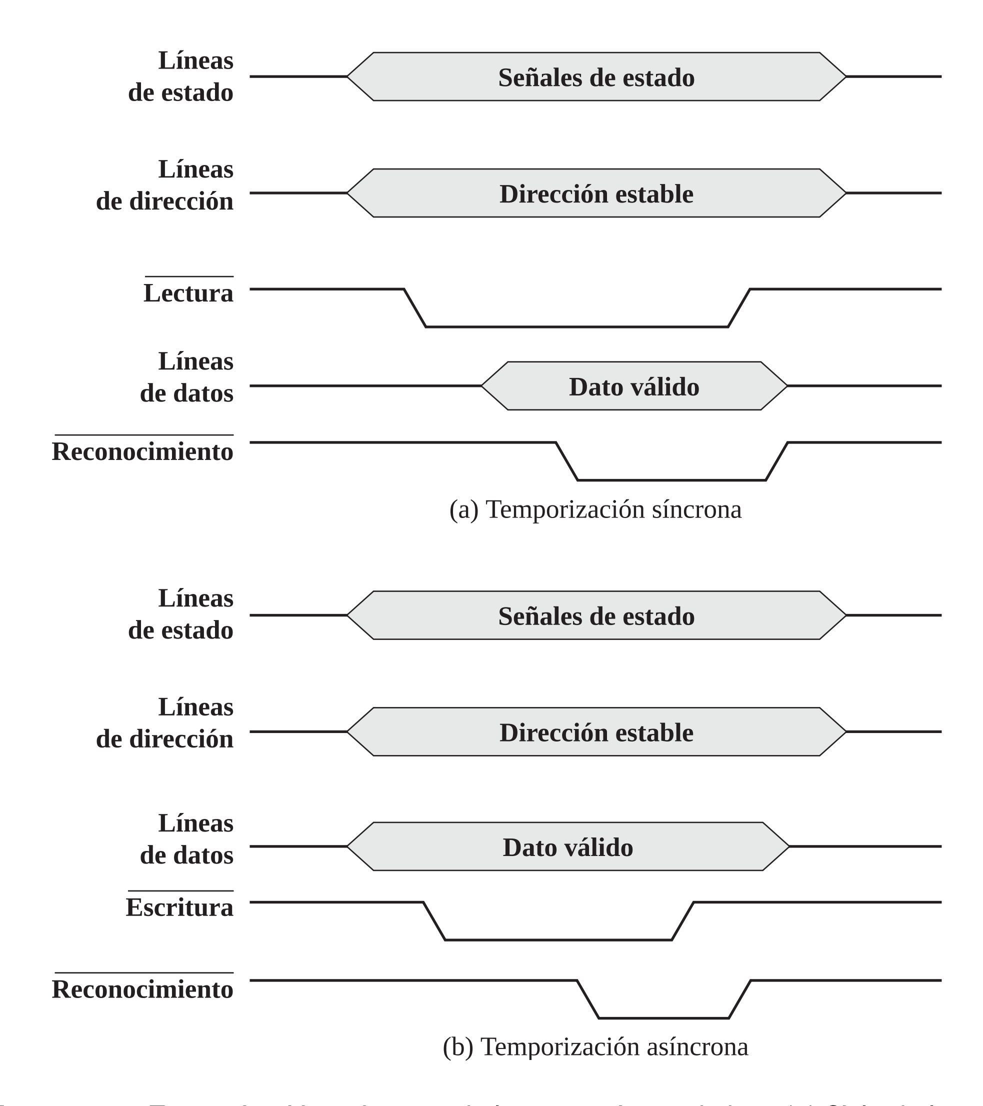


Figura 3.20. Temporización asíncrona de las operaciones de bus. (a) Ciclo de lectura del bus de sistema. (b) Ciclo de eescritura del bus de sistema.

sistema: cuanto más ancho es el bus de direcciones, mayor es el rango de posiciones a las que se puede hacer referencia.

Tipo de tranferencia de datos. Por último, un bus permite varios tipos de transferencias de datos, tal y como ilustra la Figura 3.21. Todos los buses permiten tanto transferencias de escritura (dato de maestro a esclavo) como de lectura (dato de esclavo a maestro). En el caso de un bus con direcciones y datos multiplexados, el bus se utiliza primero para especificar la dirección y luego para transferir el dato. En una operación de lectura, generalmente hay un tiempo de espera mientras el dato se está captando del dispositivo esclavo para situarlo en el bus. Tanto para la lectura como para la escritura, puede haber también un retardo si se necesita utilizar algún procedimiento de arbitraje para acceder al control del bus en el resto de la operación (es decir, tomar el bus para

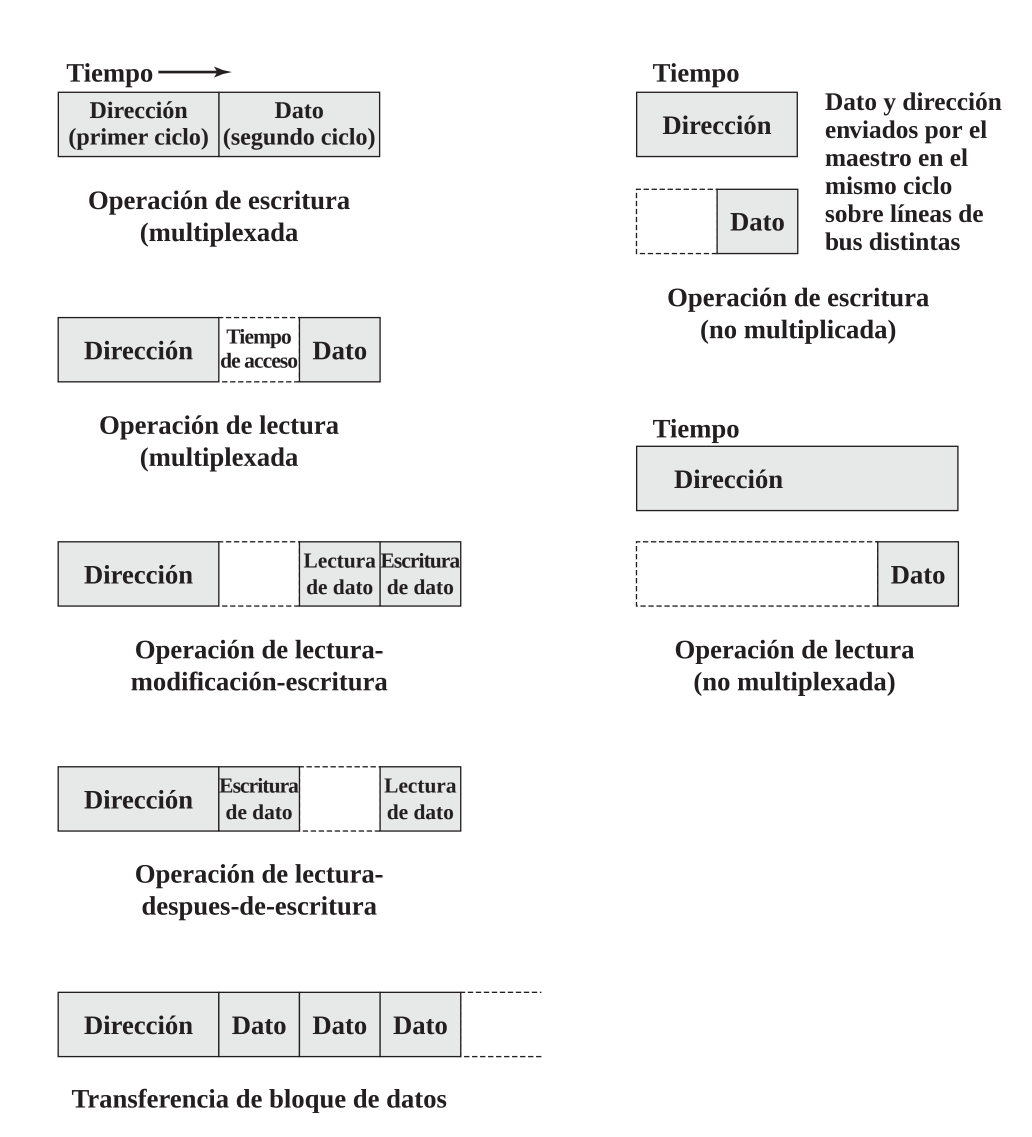


Figura 3.21. Tipos de transferencias de datos en un bus.

solicitar una lectura o una escritura, y después tomar el bus de nuevo para realizar la lectura o la escritura).

En el caso de que haya líneas dedicadas a datos y a direcciones, la dirección se sitúa en el bus de direcciones y se mantiene ahí mientras que el dato se ubica en el bus de datos. En una escritura, el maestro pone el dato en el bus de datos tan pronto como se han estabilizado las líneas de dirección y el esclavo ha podido reconocer su dirección. En una operación de lectura, el esclavo pone el dato en el bus de datos tan pronto como haya reconocido su dirección y disponga del mismo.

En ciertos buses también son posibles algunas operaciones combinadas. Una operación de lectura-modificación-escritura es simplemente una lectura seguida inmediatamente de una escritura en la misma dirección. La dirección se proporciona una sola vez al comienzo de la operación. La operación

completa es generalmente indivisible de cara a evitar cualquier acceso al dato por otros posibles maestros del bus. El objetivo primordial de esta posibilidad es proteger los recursos de memoria compartida en un sistema con multiprogramación (*véase* el Capítulo 8).

La lectura-después de-escritura es una operación indivisible que consiste en una escritura seguida inmediatamente de una lectura en la misma dirección. La operación de lectura se puede realizar con el propósito de comprobar el resultado.

Algunos buses también permiten transferencias de bloques de datos. En este caso, un ciclo de dirección viene seguido por *n* ciclos de datos. El primer dato se transfiere a o desde la dirección especificada; el resto de datos se transfieren a o desde las direcciones siguientes.

3.5. PCI

El bus PCI (*Peripheral Component Interconnect,* Interconexión de Componente Periférico) es un bus muy popular de ancho de banda elevado, independiente del procesador, que se puede utilizar como bus de periféricos o bus para una arquitectura de entreplanta. Comparado con otras especificaciones comunes de bus, el PCI proporciona mejores prestaciones para los subsistemas de E/S de alta velocidad (por ejemplo, los adaptadores de pantalla gráfica, los controladores de interfaz de red, los controladores de disco, etc.). El estándar actual permite el uso de hasta 64 líneas de datos a 66 MHz, para una velocidad de transferencia de 528 MB, o 4,224 Gbps. Pero no es precisamente su elevada velocidad la que hace atractivo al PCI. El PCI ha sido diseñado específicamente para ajustarse, económicamente a los requisitos de E/S de los sistemas actuales; se implementa con muy pocos circuitos integrados y permite que otros buses se conecten al bus PCI.

Intel empezó a trabajar en el PCI en 1990 pensando en sus sistemas basados en el Pentium. Muy pronto Intel cedió sus patentes al dominio público y promovió la creación de una asociación industrial, la PCI SIG (de *Special Interest Group*), para continuar el desarrollo y mantener la compatibilidad de las especificaciones del PCI. El resultado ha sido que el PCI ha sido ampliamente adoptado y se está incrementando su uso en los computadores personales, estaciones de trabajo, y servidores. Puesto que las especificaciones son de dominio público y están soportadas por una amplia banda de la industria de procesadores y periféricos, los productos PCI fabricados por compañías diferentes son compatibles.

El PCI está diseñado para permitir una cierta variedad de configuraciones basadas en microprocesadores, incluyendo sistemas tanto de uno como de varios procesadores. Por consiguiente, proporciona un conjunto de funciones de uso general. Utiliza temporización síncrona y un esquema de arbitraje centralizado.

La Figura 3.22a muestra la forma usual de utilizar el bus PCI en un sistema monoprocesador. Un dispositivo que integra el controlador de DRAM y el adaptador al bus PCI proporciona el acoplamiento al procesador y la posibilidad de generar datos a velocidades elevadas. El adaptador actúa como un registro de acoplo (buffer) de datos puesto que la velocidad del bus PCI puede diferir de la capacidad de E/S del procesador. En un sistema multiprocesador (Figura 3.22b), se pueden conectar mediante adaptadores una o varias configuraciones PCI al bus de sistema del procesador. Al bus de sistema se conectan únicamente las unidades procesador/caché, la memoria principal y los adaptadores de PCI. De nuevo, el uso de adaptadores mantiene al PCI independiente de la velocidad del procesador y proporciona la posibilidad de recibir y enviar datos rápidamente.

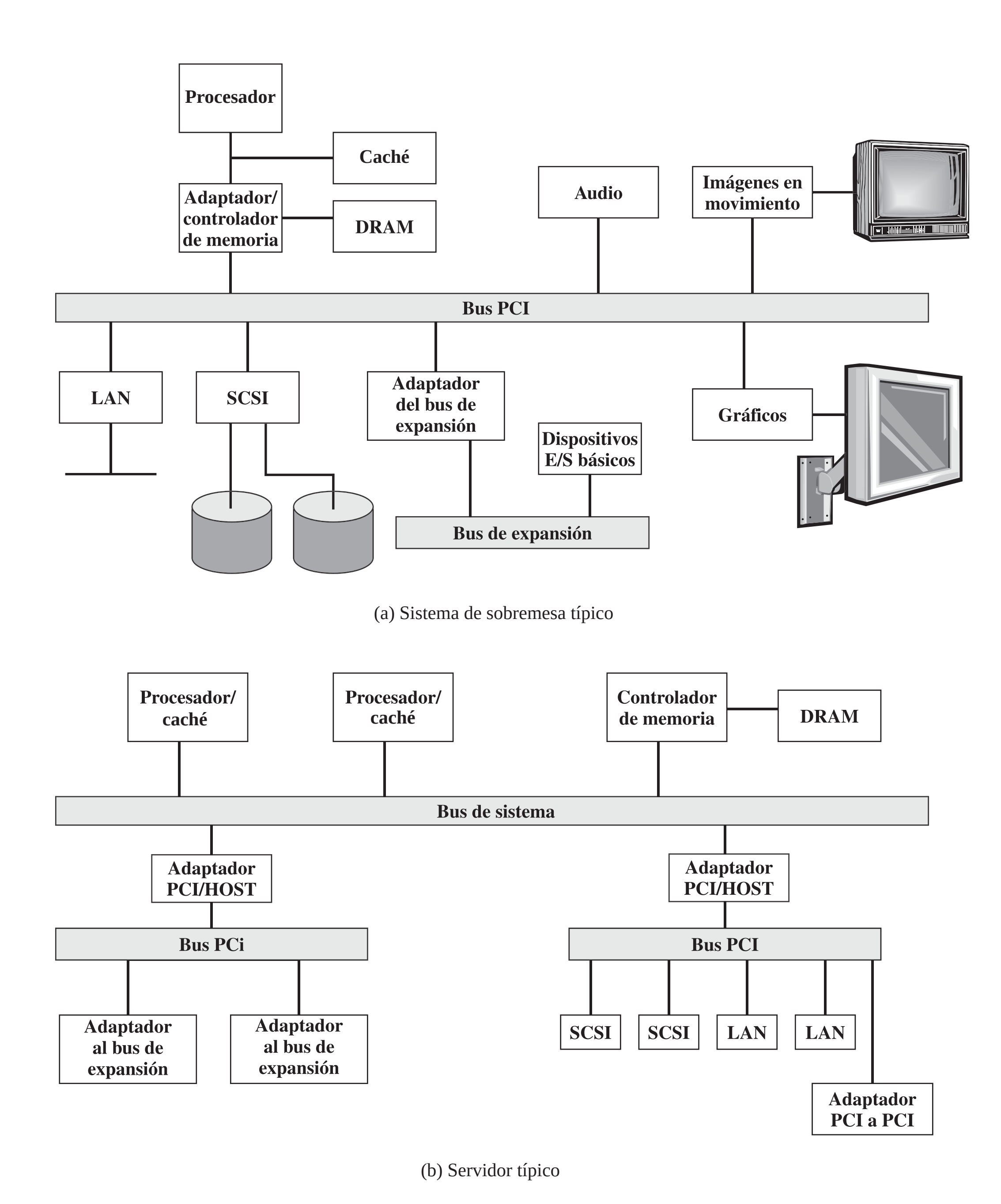


Figura 3.22. Ejemplos de configuraciones PCI.

ESTRUCTURA DEL BUS

El bus PCI puede configurarse como un bus de 32 o 64 bits. La Tabla 3.3 define las 49 líneas de señal obligatorias en el PCI. Se dividen en los grupos funcionales siguientes:

Tabla 3.3. Líneas obligatorias del bus PCI.

Denominación	Tipo	Descripción
Terminales de sistema		
CLK	in	Proporciona la temporización para todas las transacciones y es muestreada en el flanco de subida. Se pueden utilizar frecuencias de reloj de hasta 33 MHz.
RST#	in	Hace que todos los registros, secuenciadores y señales específicas de PCI pasen a su estado de inicio.
Terminales de direcciones y datos		
AD[31:0]	t/s	Líneas multiplexadas para direcciones y datos.
C/BE[3:0]#	t/s	Señales multiplexadas de órdenes del bus y de byte activo (*byte enable*). Durante la fase de datos, las líneas indican cuál de los cuatro grupos de líneas de byte transporta información válida.
PAR	t/s	Proporciona paridad para para las líneas AD y C/BE, un ciclo de reloj después. El maestro proporciona PAR para las fases de dirección y escritura de datos, y el dispositivo de lectura genera PAR en la fase de lectura de datos.
Terminales de control de interfaz		
FRAME#	s/t/s	Suministradas por el maestro actual para indicar el comienzo y la duración de una transferencia. Las activa al comienzo y las desactiva cuando el maestro está preparado para empezar el final de la fase de datos.
IRDY#	s/t/s	Iniciador preparado (*Initiator Ready*). La proporciona el maestro actual del bus (el iniciador de la transacción). Durante una lectura, indica que el maestro está preparado para aceptar datos; durante una escritura indica que el dato válido está en AD.
TRDY#	s/t/s	Dispositivo preparado (*Target Ready*). La proporciona el dispositivo seleccionado. Durante una lectura , el dato válido está en AD; durante una escritura indica que el destino está preparado para aceptar datos.
STOP#	s/t/s	Indica que el dispositivo seleccionado desea que el maestro pare la transacción actual.
LOCK#	s/t/s	Indica una operación atómica indivisible que puede requerir varias transferencias.
IDSEL#	in	Selector de inicio de dispositivo (*Initialization Device Select*). Usada como señal de selección de circuito (*chip select*) durante las lecturas y escrituras de configuración.
DEVSEL#	in	Selector de dispositivo (*Device Select*). Activada por el dispositivo seleccionado cuando ha reconocido su dirección. Indica al maestro actual si se ha seleccionado un dispositivo.
Terminales de arbitraje		
REQ#	t/s	Indica al árbitro que el dispositivo correspondiente solicita utilizar el bus. Es una línea punto-a-punto específica para cada dispositivo.
GNT#	t/s	Indica al dispositivo que el árbitro le ha cedido el acceso al bus. Es una línea punto-a-punto específica para cada dispositivo.
Terminales para señales de error		
PERR#	s/t/s	Error de paridad. Indica que se ha detectado un error de paridad en los datos por parte del dispositivo en el caso de una escritura, o por parte del maestro en el caso de una lectura.
SERR#	o/d	Error del sistema. La puede activar cualquier dispositivo para comunicar errores de paridad en la dirección u otro tipo de errores críticos distintos de la paridad.

- **Terminales («patillas») de sistema:** constituidas por los terminales de reloj y de inicio (reset).
- **Terminales de direcciones y datos:** incluye 32 líneas para datos y direcciones multiplexadas en el tiempo. Las otras líneas del grupo se utilizan para interpretar y validar las líneas de señal correspondientes a los datos y a las direcciones.
- **Terminales de control de la interfaz:** controlan la temporización de las transferencias y proporcionan coordinación entre los que las inician y los destinatarios.
- **Terminales de arbitraje:** a diferencia de las otras líneas de señal del PCI, estas no son líneas compartidas. En cambio, cada maestro del PCI tiene su par propio de líneas que lo conectan directamente al árbitro del bus PCI.
- **Terminales para señales de error:** utilizadas para indicar errores de paridad u otros.

Además, la especificación del PCI define 51 señales opcionales (Tabla 3.4), divididas en los siguientes grupos funcionales:

- **Terminales de interrupción:** para los dispositivos PCI que deben generar peticiones de servicio. Igual que los terminales de arbitraje, no son líneas compartidas sino que cada dispositivo PCI tiene su propia línea o líneas de petición de interrupción a un controlador de interrupciones.
- **Terminales de soporte de caché:** necesarios para permitir memorias caché en el bus PCI asociadas a un procesador o a otro dispositivo. Estos terminales permiten el uso de protocolos de coherencia de caché de sondeo de bus (*snoopy caché*) (en el Capítulo 16 se discuten estos protocolos).
- **Terminales de ampliación a bus de 64 bits:** incluye 32 líneas multiplexadas en el tiempo para direcciones y datos y se combinan con las líneas obligatorias de dirección y datos para constituir un bus de direcciones y datos de 64 bits. Hay otras líneas de este grupo que se utilizan para interpretar y validar las líneas de datos y direcciones. Por último, hay dos líneas que permiten que dos dispositivos PCI se pongan de acuerdo para usar los 64 bits.
- **Terminales de test (JTAG/*Boundary Scan*):** estas señales se ajustan al estándar IEEE 1149.1 para la definición de procedimientos de test.

ÓRDENES DEL PCI

La actividad del bus consiste en transferencias entre elementos conectados al bus, denominándose maestro al que inicia la transferencia. Cuando un maestro del bus adquiere el control del mismo, determina el tipo de transferencia que se producirá a continuación. Durante la fase de direccionamiento de transferencia, se utilizan las líneas C/BE para indicar el tipo de transferencia. Los tipos de órdenes son:

- Reconocimiento de interrupción
- Ciclo especial
- Lectura de E/S
- Escritura en E/S
- Lectura de memoria

Tabla 3.4. Líneas opcionales del bus PCI.

Denominación	Tipo	Descripción
Terminales de interrupción		
INTA#	o/d	Utilizada para pedir una interrupción.
INTB#	o/d	Utilizada para pedir una interrupción; solo tiene significado en un dispositivo multifunción.
INTC#	o/d	Utilizada para pedir una interrupción; solo tiene significado en un dispositivo multifunción.
INTD#	o/d	Utilizada para pedir una interrupción; solo tiene significado en un dispositivo multifunción.
Terminales de soporte de caché		
SBO#	in/out	(*Snoop Backoff*) Indica un acceso a una línea de caché modificada.
SDONE	in/out	(*Snoop Done*) Indica el estado del módulo de sondeo del bus (*snoop*) en el acceso actual a caché. Se activa cuando el sondeo ha terminado.
Terminales de extensión a bus de 64 bits		
AD[63:32]	t/s	Líneas multiplexadas de direcciones y datos para ampliar el bus a 64 bits.
C/BE[7:4]	t/s	Líneas multiplexadas de órdenes y habilitación de byte. Durante la fase de dirección, las líneas proporcionan las órdenes adicionales del bus. Durante la fase de datos, las líneas indican cual de los cuatro bytes del bus ampliado contiene datos válidos.
REQ64#	s/t/s	Utilizada para pedir una transferencia de 64 bits.
ACK64#	s/t/s	Indica que el dispositivo seleccionado está dispuesto para realizar una transferencia de 64 bits.
PAR64	t/s	Proporciona paridad par para las líneas ampliadas AD y para C/BE un ciclo de reloj después.
Terminales de test (JTAG/*Boundary Scan*)		
TCK	in	Reloj de test. Utilizado para sincronizar la entrada y salida de la información de estado y los datos de test del dispositivo testeado (mediante *Boundary Scan*).
TDI	in	Entrada de test. Utilizada para introducir bit a bit los datos y las instrucciones de test en el dispositivo a testear.
TDO	out	Salida de test. Utilizada para obtener bit a bit los datos y las instrucciones de test desde el dispositivo testeado.
TMS	in	Selector de modo de test. Utilizado para establecer el estado del controlador del puerto de acceso para el test.
TRS#	in	Inicicio de test (test reset). Utilizada para iniciar el controlador del puerto de acceso para el test.

in = Señal de entrada
out = Señal de salida
t/s = Señal de E/S, bidireccional, tri-estado
s/t/s = Señal tri-estado activada solo por un dispositivo en cada momento
o/d = Drenador abierto: permite que varios dispositivos lo compartan como en una OR cableada
= La señal se activa en el nivel inferior de tensión (activa en baja)

- Lectura de línea de memoria
- Lectura múltiple de memoria
- Escritura en memoria
- Escritura e invalidación de memoria
- Lectura de configuración
- Escritura de configuración
- Ciclo de dirección dual

El Reconocimineto de Interrupción es una orden de lectura proporcionada por el dispositivo que actúa como controlador de interrupciones en el bus PCI. Las líneas de direcciones no se utilizan en la fase de direccionamiento, y las líneas de byte activo (*byte enable*) indican el tamaño del identificador de interrupción a devolver.

La orden de ciclo especial se utiliza para iniciar la difusión de un mensaje a uno o más destinos.

Las órdenes de lectura de E/S y escritura en E/S se utilizan para intercambiar datos entre el módulo que inicia la transferencia y un controlador de E/S. Cada dispositivo de E/S tiene su propio espacio de direcciones y las líneas de direcciones se utilizan para indicar un dispositivo concreto y para especificar los datos a trasferir a o desde ese dispositivo. El concepto de direcciones de E/S se explora en el Capítulo 7.

Las órdenes de lectura y escritura en memoria se utilizan para especificar la transferencia de una secuencia de datos, utilizando uno o más ciclos de reloj. La interpretación de estas órdenes depende de si el controlador de memoria del bus PCI utiliza el protocolo PCI para transferencias entre memoria y caché, o no. Si lo utiliza, la transferencia de datos a y desde la memoria normalmente se produce en términos de líneas o bloques de caché [2]. El uso de las tres órdenes de lectura de memoria se resume en la Tabla 3.5. La orden de escritura en memoria se utiliza para transferir datos a memoria

Tabla 3.5 Interpretación de las órdenes de Lectura del bus PCI.

Tipo de orden de lectura	Para memoria transferible a caché	Para memoria no transferible a caché
Lectura de memoria	Secuencia de la mitad o menos de una línea	Secuencia de dos o menos ciclos de transferencia de datos
Lectura de línea de memoria	Secuencia de más de media de línea y menos de tres líneas de caché	Secuencia de tres a doce ciclos de transferencia
Lectura de memoria múltiple	Secuencia de más de tres líneas de caché	Secuencia de más de doce ciclos de transferencia de datos

[2] Los principios fundamentales de la memoria caché se describe en el Capítulo 4; los protocolos de caché basados en buses se describen en el Capítulo 18.

en uno o más ciclos de datos. La orden de escritura e invalidación de memoria transfiere datos a memoria en uno o más ciclos. Además, indica que al menos se ha escrito en una línea de caché. Esta orden permite el funcionamiento de la caché con postescritura (*write back*) en memoria.

Las dos órdenes de configuración permiten que un dispositivo maestro lea y actualice los parámetros de configuración de un dispositivo conectado al bus PCI. Cada dispositivo PCI puede disponer de hasta 256 registros internos utilizados para configurar dicho dispositivo durante la inicialización del sistema.

La orden de ciclo de dirección dual se utiliza por el dispositivo que inicia la transferencia para indicar que está utilizando direcciones de 64 bits.

TRANSFERENCIAS DE DATOS

Toda transferencia de datos en el bus PCI es una transacción única que consta de una fase de direccionamiento y una o más fases de datos. En esta discusión, se ilustra una operación de lectura típica; las operaciones de escritura se producen de forma análoga.

La Figura 3.23 muestra la temporización de una operación de lectura. Todos los eventos se sincronizan en las transiciones de bajada del reloj, cosa que sucede a la mitad de cada ciclo de reloj. Los dispositivos del bus interpretan las líneas del bus en los flancos de subida al comienzo del ciclo de bus. A continuación se describen los eventos significativos señalados en el diagrama:

a) Una vez que el maestro del bus ha obtenido el control del bus, debe iniciar la transacción activando FRAME. Esta línea permanece activa hasta que el maestro está dispuesto para terminar la última fase de datos. El maestro también sitúa la dirección de inicio en el bus de direcciones y la orden de lectura en las líneas C/BE.

b) Al comienzo del ciclo de reloj 2, el dispositivo del que se lee reconocerá su dirección en las líneas AD.

c) El maestro deja libres las líneas AD del bus. En todas las líneas de señal que pueden ser activadas por más de un dispositivo se necesita un ciclo de cambio (indicado por las dos flechas circulares) para que la liberación de las líneas de dirección permita que el bus pueda ser utilizado por el dispositivo de lectura. El maestro cambia la información de las líneas C/BE para indicar cuales de las líneas AD se utilizan para transferir el dato direccionado (de 1 a 4 bytes). El maestro también activa IRDY para indicar que está preparado para recibir el primer dato.

d) El dispositivo de lectura seleccionado activa DEVSEL para indicar que ha reconocido las direcciones y va a responder. Sitúa el dato solicitado en las líneas AD y activa TRDY para indicar que hay un dato válido en el bus.

e) El maestro lee el dato al comienzo del ciclo de reloj 4 y cambia las líneas de habilitación de byte según se necesite para la próxima lectura.

f) En este ejemplo, el dispositivo de lectura necesita algún tiempo para preparar el segundo bloque de datos para la transmisión. Por consiguiente, desactiva TRDY para señalar al maestro que no proporcionará un nuevo dato en el próximo ciclo. En consecuencia, el maestro no lee las líneas de datos al comienzo del quinto ciclo de reloj y no cambia la

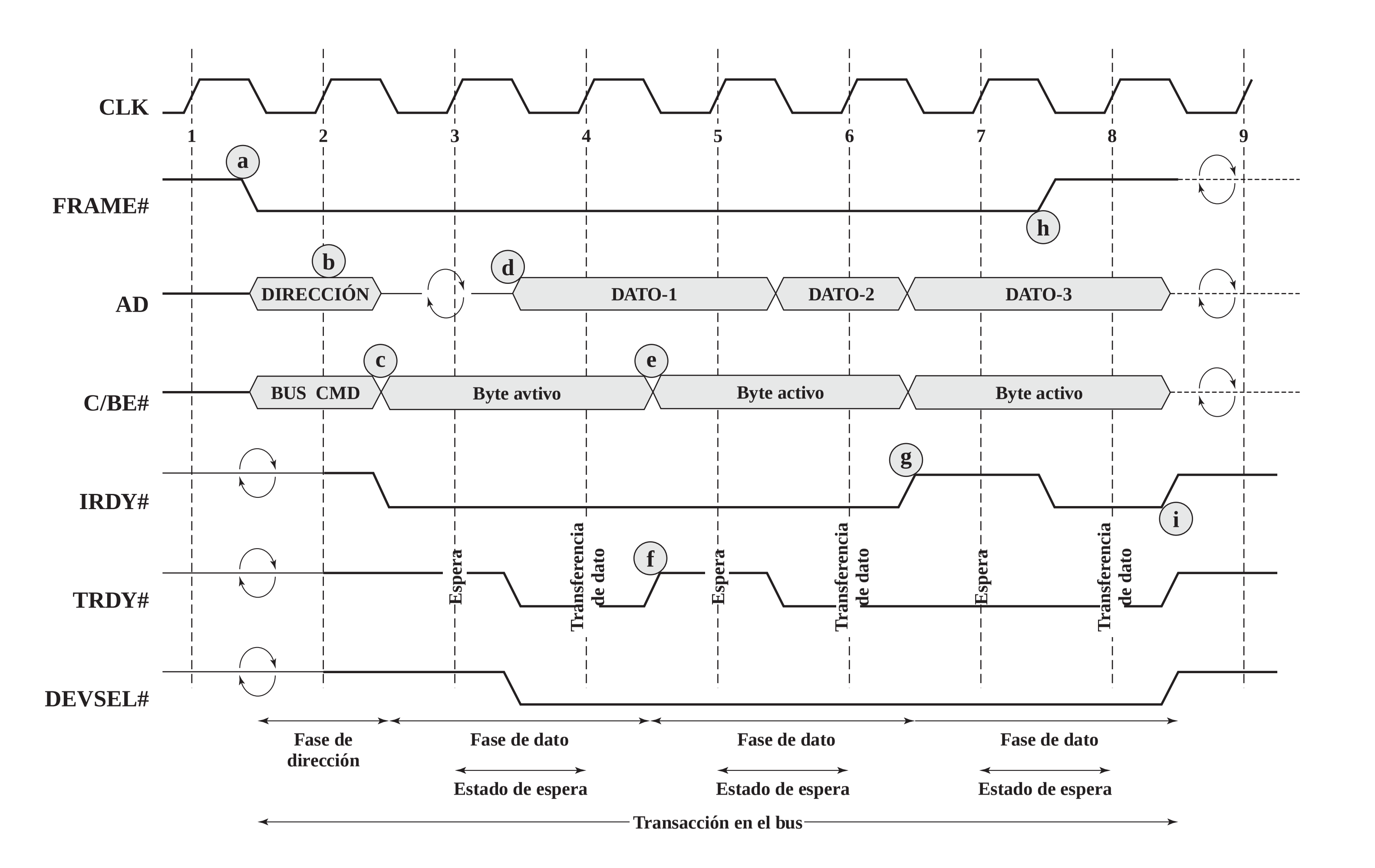


Figura 3.23. Operación de lectura PCI.

señal de habilitación de byte durante ese ciclo. El bloque de datos es leído al comienzo del ciclo de reloj 6.

g) Durante el ciclo 6, el dispositivo de lectura sitúa el tercer dato en el bus. No obstante, en este ejemplo, el maestro todavía no está preparado para leer el dato (por ejemplo, puede tener lleno el registro de almacenamiento temporal). Para indicarlo desactiva IRDY. Esto hará que el dispositivo de lectura mantenga el tercer dato en el bus durante un ciclo de reloj extra.

h) El maestro sabe que el tercer dato es el último, y por eso desactiva FRAME para indicar al dispositivo de lectura que este es el último dato a transferir. Además activa IRDY para indicar que está listo para completar esa transferencia.

i) El maestro desactiva IRDY, haciendo que el bus vuelva a estar libre, y el dispositivo de lectura desactiva TRDY y DEVSEL.

ARBITRAJE

El bus PCI utiliza un esquema de arbitraje centralizado síncrono en el que cada maestro tiene una única señal de petición (REQ) y cesión (GNT) del bus. Estas líneas se conectan a un árbitro central (Figura 3.24) y se utiliza un simple intercambio de las señales de petición y cesión para permitir el acceso al bus.

La especificación PCI no indica un algoritmo particular de arbitraje. El árbitro puede utilizar un procedimiento de primero-en llegar-primero-en servirse, un procedimiento de cesión cíclica (*round-robin*), o cualquier clase de esquema de prioridad. El maestro del PCI establece, para cada transferencia que desee hacer, si tras la fase de dirección sigue una o más fases de datos consecutivas.

La Figura 3.25 es un ejemplo en el que se decide el dispositivo, A o B, al que se cede el bus. Se produce la siguiente secuencia:

a) En algún momento anterior al comienzo del ciclo de reloj 1, A ha activado su señal REQ. El árbitro muestrea esa señal al comienzo del ciclo de reloj 1.

b) Durante el ciclo de reloj 1, B solicita el uso del bus activando su señal REQ.

c) Al mismo tiempo, el árbitro activa GNT-A para ceder el acceso al bus a A.

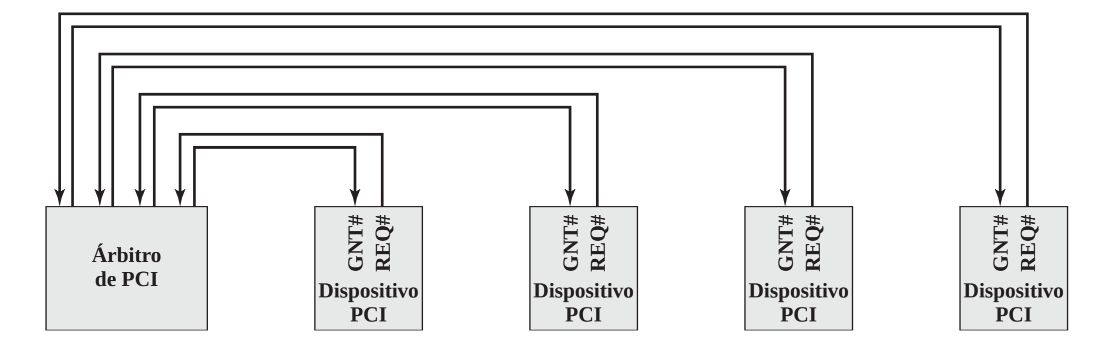


Figura 3.24. Árbitro de bus PCI.

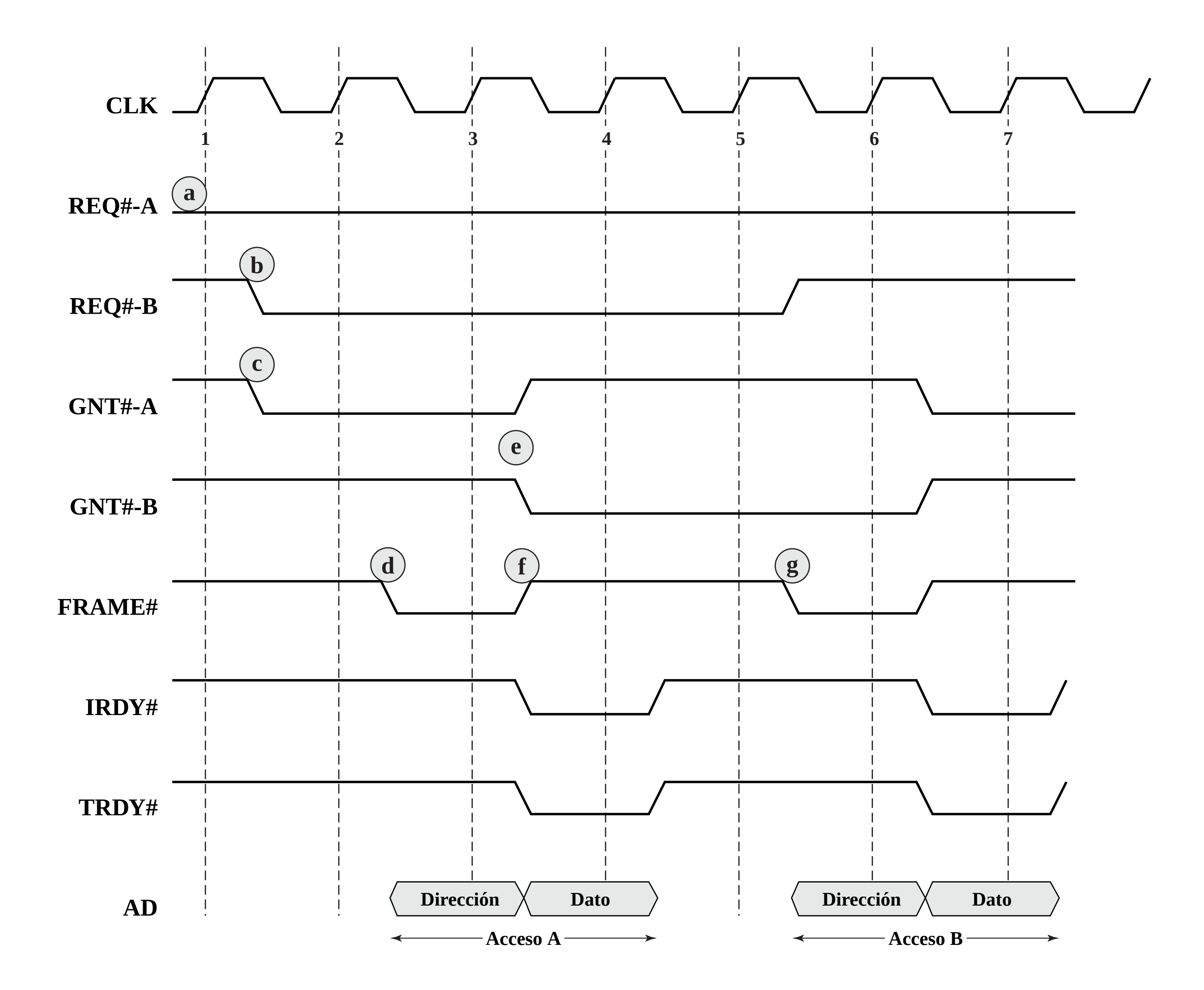


Figura 3.25. Arbitraje del bus PCI entre dos maestros.

d) El maestro del bus A muestrea GNT-A al comienzo del ciclo de reloj 2 y conoce que se le ha cedido el acceso al bus. Además, encuentra IRDY y TRDY desactivados, indicando que el bus está libre. En consecuencia, activa FRAME y coloca la información de dirección en el bus de direcciones y la orden correspondiente en las líneas C/BE (no mostradas). Además mantiene activa REQ-A, puesto que tiene que realizar otra transferencia después de la actual.

e) El árbitro del bus muestrea todas las líneas REQ al comienzo del ciclo 3 y toma la decisión de ceder el bus a B para la siguiente transacción. Entonces activa GNT-B y desactiva GNT-A. B no podrá utilizar el bus hasta que este no vuelva a estar libre.

f) A desactiva FRAME para indicar que la última (y la única) transferencia de dato está en marcha. Pone los datos en el bus de datos y se lo indica al dispositivo destino con IRDY. El dispositivo lee el dato al comienzo del siguiente ciclo de reloj.

g) Al comienzo del ciclo 5, B encuentra IRDY y FRAME desactivados y por consiguiente puede tomar el control del bus activando FRAME. Además desactiva su línea REQ, puesto que solo deseaba realizar una transferencia.

Posteriormente, se cede el acceso al bus al maestro A para que realice su siguiente transferencia.

Hay que resaltar que el arbitraje se produce al mismo tiempo que el maestro del bus actual está realizando una transferencia de datos. Por consiguiente, no se pierden ciclos de bus en realizar el arbitraje. Esto se conoce como *arbitraje oculto o solapado (hidden arbitration)***.**

3.6. LECTURAS Y SITIOS WEB RECOMENDADOS

El libro con la descripción más clara de PCI es [SHAN99]. [ABBO04] también contiene bastante información rigurosa de PCI.

ABBO04 Abbot, D.: *PCI Bus Desmystified.* New York. Elsevier, 2004.

SHAN99 Shanley, T., y Anderson, D.: *PCI Systems Architecture.* Richardson, TX. Mindshare Press, 1999.



SITIOS WEB RECOMENDADOS

- **PCI Special Interest Group:** información acerca de las especificaciones del bus PCI y productos basados en el mismo
- **PCI Pointers:** enlaces a suministradores y otras fuentes de información sobre PCI.

3.7. PALABRAS CLAVE, CUESTIONES Y PROBLEMAS

PALABRAS CLAVE

anchura del bus	captación de instrucción	interrupción inhabilitada
arbitraje centralizado	ciclo de instrucción	registro de datos de memoria (MBR)
arbitraje del bus	ejecución de instrucción	registro de dirección de memoria (MAR)
arbitraje distribuido	gestor de interrupciones	rutina de servicio de interrupción
bus	interconexión de componente periférico (*peripheral component interconect*, PCI)	temporización asíncrona
bus de datos	interrupción	temporización síncrona
bus de direcciones		
bus de sistema		

CUESTIONES

3.1. ¿Qué tipos generales de funciones especifican las instrucciones de un computador?

3.2. Enumere y defina brevemente los estados posibles que determinan la ejecución de una instrucción.

3.3. Enumere y defina brevemente dos aproximaciones para gestionar las instrucciones múltiples.

3.4. ¿Qué tipos de transferencias debe permitir la estructura de interconexión (por ejemplo, un bus) de un computador?

3.5. ¿Qué ventajas tiene una arquitectura de varios buses frente a otra de bus único?

3.6. Enumere y defina brevemente los grupos de líneas de señal para el bus PCI.

PROBLEMAS

3.1. La máquina hipotética de la Figura 3.4 también tiene dos instrucciones de E/S:

0011 = Cargar AC desde E/S
0111 = Almacenar AC en E/S

En estos casos, la dirección de 12 bits identifica un dispositivo concreto de E/S. Muestre la ejecución del programa (utilizando el formato de la Figura 3.5) para el siguiente programa:

1. Cargar AC desde el dispositivo 5.
2. Sumar el contenido de la posición de memoria 940.
3. Almacenar AC en el dispositivo 6.

Asuma que el siguiente valor obtenido desde el dispositivo 5 es 3 y que la posición 940 almacena el valor 2.

3.2. La ejecución de un programa que se proporciona en la Figura 3.5 se describe en el texto utilizando seis etapas. Amplíe esta descripción indicando la forma en que se utilizan los registros MAR y MBR.

3.3. Considere un hipotético microprocesador de 32 bits cuyas instrucciones de 32 bits están compuestas por dos campos: el primer byte contiene el código de operación (codop) y los restantes un operando inmediato o una dirección de operando.

(a) ¿Cuál es la máxima capacidad de memoria (en bytes) direccionable directamente?
(b) Discuta el impacto que se produciría en la velocidad del sistema si el microprocesador tiene:
   1. Un bus de dirección local de 32 bits y un bus de datos local de 16 bits, o
   2. un bus de dirección local de 16 bits y un bus de datos local de 16 bits.
(c) ¿Cuántos bits necesitan el contador de programa y el registro de instrucción?

3.4. Considere un microprocesador hipotético que genera direcciones de 16 bits (por ejemplo, suponga que el contador de programa y el registro de dirección son de 16 bits) y tiene un bus de datos de 16 bits.

(a) ¿Cuál es el máximo espacio de direcciones de memoria al que el procesador puede acceder directamente si está conectado a una «memoria de 16 bits»?
(b) ¿Cuál es el máximo espacio de direcciones de memoria al que el procesador puede acceder directamente si está conectado a una «memoria de 8 bits»?
(c) ¿Qué características de la arquitectura permitirán a este procesador acceder a un «espacio de E/S» separado?
(d) Si una instrucción de entrada o de salida pueden especificar un número de puerto de E/S de 8 bits, ¿cuántos puertos de E/S de 8 bits puede soportar el microprocesador? (cuántos puertos de E/S de 16 bits? Explíquelo.

3.5. Considere un microprocesador de 32 bits, con un bus externo de 16 bits, y con una entrada de reloj de 8 MHz. Asuma que el procesador tiene un ciclo de bus cuya duración mínima es igual a cuatro ciclos de reloj. ¿Cuál es la velocidad de transferencia máxima que puede sostener el microprocesador? Para

incrementar sus prestaciones, ¿sería mejor hacer que su bus externo de datos sea de 32 bits o doblar la frecuencia de reloj que se suministra al microprocesador? Establezca las suposiciones que considere y explíquelo. *Ayuda:* determine el número de bytes que pueden transferirse por ciclo de bus.

3.6. Considere un computador que posee un módulo de E/S controlando un teletipo con teclado e impresora. La CPU tiene los siguientes registros conectados directamente al bus de sistema:

INPR:	Registro de Entrada - 8 bits
OUTPR:	Registro de Salida - 8 bits
FGI:	Indicador (Flag) de Entrada - 1 bit
FGO:	Indicador (Flag) de Salida - 1 bit
IEN:	Habilitación de Interrupción - 1 bit

La entrada desde el teclado y la salida a la impresora del teletipo están controlados por el módulo de E/S. El teletipo es capaz de codificar un símbolo alfanumérico mediante una palabra de 8 bits y decodificar una palabra de 8 bits en un símbolo alfanumérico.

(a) Describa cómo el procesador, utilizando el primero de los cuatro registros enumerados anteriormente, puede realizar una E/S con el teletipo.

(b) Describa cómo se puede realizar dicha operación de manera más eficiente si además se utiliza IEN.

3.7. Considere dos microprocesadores con buses de datos externos de 8 y 16 bits, respectivamente. Los dos procesadores son idénticos en todo lo demás y sus ciclos de bus son iguales.

(a) Suponiendo que todas las instrucciones y operandos son de dos bytes, ¿en qué factor difieren las velocidades de transferencia de los dos microprocesadores?

(b) ¿cuál sería la respuesta si en la mitad de los casos los operandos y las instrucciones son de un byte?

3.8. La Figura 3.26 muestra un esquema de arbitraje distribuido que puede utilizarse con un bus actualmente obsoleto denominado Multibus I. Los módulos se conectan en cadena (*daisy-chain*) según su orden de prioridad. El módulo que está más a la izquierda en el diagrama recibe una *prioridad de bus* constante a través de la señal BPRN indicando que ningún módulo de mayor prioridad solicita el bus. Si el módulo no desea utilizar el bus activa su línea de *salida de prioridad del bus (BPRO)***.** Al comienzo de un ciclo de reloj, cualquier módulo puede pedir el control del bus poniendo en baja su línea BPRO. Esto hace que pase a baja la línea BPRN del siguiente módulo de la cadena, que a su vez debe poner a baja su línea BPRO. Así, la señal se propaga a lo largo de la cadena. Al final de esta respuesta en cadena, únicamente hay un módulo cuya BPRN está activada y cuya BPRO no. Este módulo es el que tiene prioridad. Si, al comienzo de un ciclo de bus, el bus no está ocupado (BUSY no activada), el módulo que tiene prioridad puede tomar el control del bus activando la señal BUSY.

Se necesita un cierto intervalo de tiempo para que la señal BPR se propague desde el módulo de prioridad más alta hasta el de más baja. ¿Debe este tiempo ser menor que el ciclo de reloj? Explíquelo.

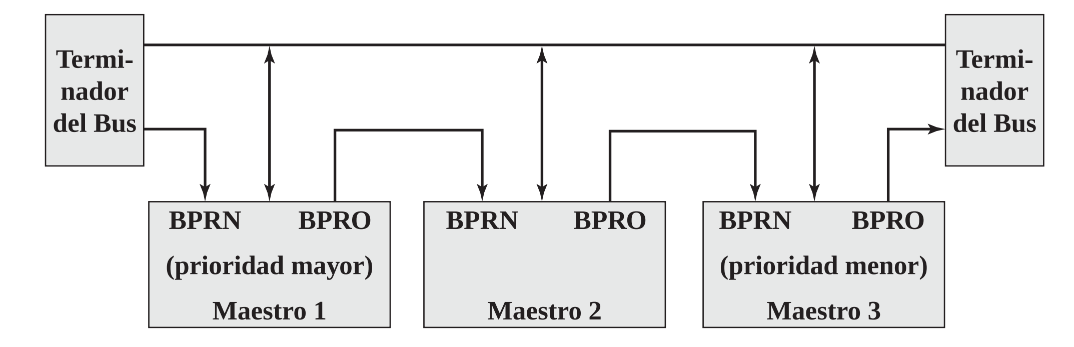


Figura 3.26. Arbitraje distribuido en el Multibus I.

3.9. El bus VAX SBI utiliza un esquema de arbitraje distribuido síncrono. Cada dispositivo SBI (es decir, procesador, memoria, módulo de E/S) tiene una prioridad única y se le asigna una única línea de solicitud de transferencia (TR, de *Transfer Request*). El SBI tiene 16 de dichas líneas (TR0,TR1...TR15), siendo TR0 la de prioridad más elevada. Cuando un dispositivo quiere utilizar el bus, realiza una reserva de un ciclo de bus futuro activando su línea TR durante el ciclo de bus en curso. Al final del ciclo en curso, cada dispositivo con una reserva pendiente examina las líneas TR; el que tiene más alta prioridad utiliza el siguiente ciclo de bus.

Como máximo se pueden conectar al bus 17 dispositivos. El dispositivo con prioridad 16 no tiene línea TR. ¿Por qué?

3.10. En el bus VAX SBI el dispositivo de prioridad más baja usualmente tiene el tiempo de espera medio más bajo. Por esta razón, a la CPU se le da usualmente la prioridad más baja en el SBI. ¿Por qué el dispositivo de prioridad 16 tiene normalmente el menor tiempo de espera medio? ¿En qué circunstancias esto no sería cierto?

3.11. En una operación de lectura síncrona (Figura 3.19), el módulo de memoria debe situar el dato en el bus con la suficiente antelación al flanco de bajada de la señal de lectura para asegurar que las señales se han estabilizado. Si el bus de un microprocesador utiliza un bus de 10 MHz y la señal de lectura empieza a caer en la segunda mitad de T_3:

(a) Determine la longitud del ciclo de instrucción de una lectura de memoria.

(b) ¿Cuándo debería situarse el dato de memoria en el bus como muy tarde? Suponga un tiempo de 20 ns para la estabilización de las líneas de datos.

3.12, Considere un microprocesador con la temporización de lectura de memoria de la Figura 3.19. Después de analizar la situación, un diseñador determina que la memoria no es capaz de proporcionar el dato a leer en un tiempo inferior a 180 ns.

(a) ¿Cuántos estados de espera (ciclos de reloj) deben insertarse para conseguir una lectura adecuada de memoria si el reloj del bus es de 8 MHz?

(b) Para utilizar los estados de espera se añade una línea de estado denominada «Ready». Una vez que el procesador ha activado la señal de lectura debe esperar a que la línea «Ready» se active para intentar leer el dato de las líneas del bus. ¿Durante qué intervalo de tiempo hay que mantener la señal «Ready» en baja para que el procesador inserte los ciclos de espera necesarios?

3.13. Un microprocesador utiliza la temporización que se muestra en la Figura 3.19 para la la escritura en memoria. Su fabricante especifica que la extensión temporal de la señal de escritura viene dada por T-50, donde T es el ciclo de reloj en nanosegundos.

(a) ¿Cuál es la extensión de la señal de escritura si la frecuencia de reloj del bus es de 5 MHz?

(b) La hoja de características del microprocesador especifica que, tras el flanco de bajada de la señal de escritura, los datos se mantienen válidos durante 20 ns. ¿Durante cuánto tiempo se mantienen válidos los datos que se van a escribir en memoria?

(c) ¿Cuántos estados de espera deberían insertarse si la memoria necesita que los datos se mantengan válidos durante al menos 190 ns.?

3.14. Un microprocesador tiene una instrucción de incremento de memoria directo que suma 1 al valor almacenado en una posición de memoria. La instrucción tiene cinco etapas: captación del código de operación (4 ciclos de reloj), captación de la dirección del operando (3 ciclos), captación del operando (3 ciclos), suma de 1 al operando (3 ciclos), y almacenar el operando (3 ciclos).

(a) ¿En qué porcentaje se incrementa la direción de la instrucción si hemos de insertar dos ciclos de bus de espera en cada operación de lectura o escritura de memoria?

(b) Repita el apartado anterior para el caso en que la operación de incremento necesite 13 ciclos en lugar de 3.

3.15. El microprocesador 8088 de Intel tiene una temporización de bus para la lectura similar a la de la Figura 3.19, pero necesita cuatro ciclos de reloj del procesador. El dato se mantiene válido en el bus

durante un tiempo que se extiende hasta el cuarto ciclo de reloj del procesador. Considere que la frecuencia de reloj del procesador es de 8 MHz.

(a) ¿Cuál es la velocidad máxima de transferencia de datos?
(b) ¿Y en el caso de que haya que insertar un estado de espera por byte transferido?

3.16. El microprocesador 8086 de Intel es un procesador similar en muchos aspectos al microprocesador de 8 bits 8088. El 8086 utiliza un bus de 16 bits que puede transferir 2 bytes a la vez, siendo par la dirección del byte menos significativo. Sin embargo, el 8086 permite palabras de operandos alineadas tanto en direcciones pares (*even-aligned*) como impares (*odd-aligned*). Si se hace referencia a una palabra alineada en una dirección impar se necesitan dos ciclos, cada uno de cuatro ciclos de reloj de bus, para transferir la palabra.Considere una instrucción del 8086 que utiliza dos operandos de 16 bits. ¿Cuánto tiempo se tarda en captar los dos operandos según las distintas posibilidades? Considere que la frecuencia de reloj es de 4 MHz y no hay estados de espera.

3.17. Considere un microprocesador de 32 bits cuyo ciclo de bus tiene la misma duración que el de un procesador de 16 bits. Asuma que, en promedio, el 20 por ciento de los operandos e instrucciones son de 32 bits, el 40 por ciento son de 16 bits, y el 40 por ciento son de solo 8 bits. Calcule la mejora que se consigue en la captación de instrucciones y operandos con el microprocesador de 32 bits.

3.18. El microprocesador del problema 3.14 inicia una etapa de captación de operando en la instrucción de incremento de memoria directo al mismo tiempo que el teclado activa una línea de petición de interrupción. ¿Después de cuánto tiempo entra el procesador en el ciclo de procesamiento de interrupción? Considere una frecuencia de reloj de 10 MHz para el bus.

3.19. Dibuje y explique un diagrama de tiempos para una operación de escritura en un bus PCI (similar a la Figura 3.23).

APÉNDICE 3A. DIAGRAMAS DE TIEMPO

En este capítulo, los diagramas de tiempo se utilizan para ilustrar las secuencias de eventos y las dependencias entre eventos. Para el lector que no esté familiarizado con los diagramas de tiempo, este apéndice proporciona una breve explicación.

La comunicación entre los dispositivos conectados a un bus se produce a través de un conjunto de líneas capaces de transmitir señales. Se pueden transmitir dos niveles diferentes de señal (niveles de tensión), representando un 0 binario y un 1 binario. Un diagrama de tiempo muestra el nivel de la señal en una línea en función del tiempo (Figura 3.27a). Por convención, el nivel 1 de la señal binaria se representa por un nivel más alto que el nivel binario 0. Usualmente, el 0 binario es el valor por defecto (o implícito). Es decir, si no se está transmitiendo un dato ni ninguna otra señal, entonces el nivel en la línea es el que representa al 0 binario. Una transición de 0 a 1 en la señal se denomina frecuentemente *flanco de subida de la señal (leading edge)*; una transición de 1 a 0 se denomina *flanco de bajada (trailing edge)*. Por claridad, a menudo las transiciones en las señales se dibujan como si ocurrieran instantáneamente. De hecho, una transición requiere un tiempo no nulo, pero este tiempo de transición es usualmente pequeño comparado con la duración de un nivel en la señal. En un diagrama de tiempos, puede ocurrir que transcurra una cantidad de tiempo variable, o en todo caso irrelevante, entre dos eventos de interés. Esto se representa con un corte en la línea de tiempo.

A veces, la señales se representan agrupadas (Figura 3.27b). Por ejemplo, si se tranfiere un dato de un byte cada vez, se necesitan ocho líneas. Generalmente, no es esencial conocer el valor exacto que se transfiere en dicho grupo sino más bien si las señales están presentes o no.

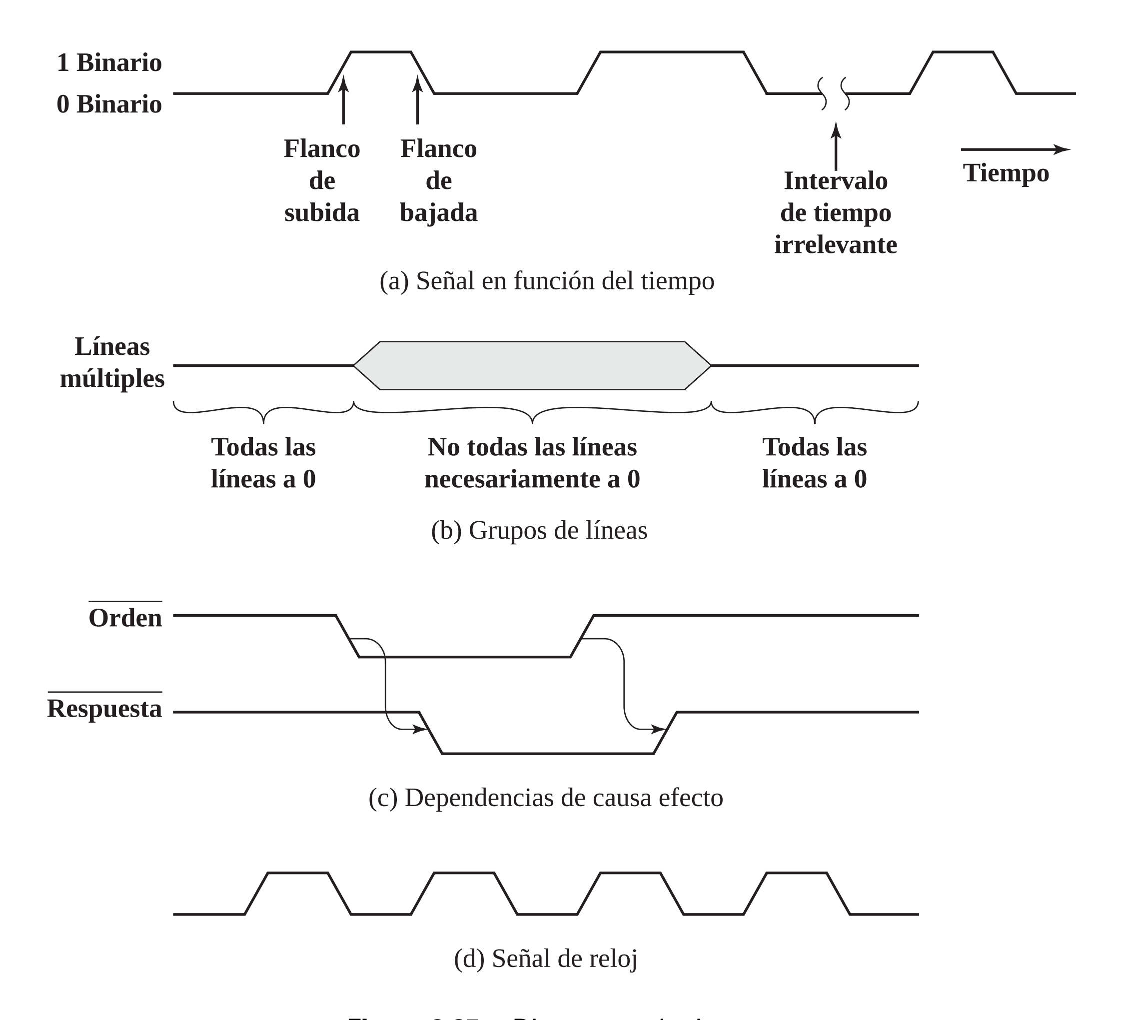


Figura 3.27. Diagramas de tiempo.

Una transición de señal en una línea provoca que un dispositivo conectado ocasione cambios de señal en otras líneas. Por ejemplo, si un módulo de memoria detecta una señal de control de lectura (transición a 0 o a 1), situará las señales correspondientes a los datos en las líneas de datos. Estas relaciones de causa-efecto dan lugar a secuencias de sucesos. Las flechas se utilizan para mostrar estas dependencias en los diagramas de tiempo (Figura 3.27c).

En la Figura 3.27c, una línea sobre el nombre de la señal indica que la señal está activa cuando está en baja. Por ejemplo, $\overline{\textit{orden}}$ está activa a 0 voltios. Esto significa que $\overline{\textit{orden}} = 0$ se interpreta como un 1 lógico.

A menudo el bus de sistema contiene una línea de reloj. Un reloj electrónico se conecta a la línea de reloj y proporciona una secuencia repetitiva y repetitiva de transiciones (Figura 3.27d). Puede haber otros eventos sincronizados con la señal de reloj.

CAPÍTULO 4

Memoria caché

PUNTOS CLAVE

- ➤ La memoria de un computador tiene una organización jerárquica. En el nivel superior (el más próximo al procesador) están los registros del procesador. A continuación se encuentran uno o más niveles de caché, denominados L1, L2, etc. Posteriormente la memoria principal, normalmente construida con memorias dinámicas de acceso aleatorio (DRAM). Todas ellas se consideran memorias internas del computador. La jerarquía prosigue con la memoria externa, siendo el siguiente nivel usualmente un disco duro fijo, y uno o más niveles de soportes extraíbles tales como discos ópticos y cintas magnéticas.
- ➤ A medida que descendemos en la jerarquía de memoria disminuye el coste por bit, aumenta la capacidad y crece el tiempo de acceso. Sería deseable poder utilizar solo la memoria más rápida, pero al ser la más costosa se llega a un compromiso entre tiempo de acceso y coste, empleando más cantidad de memoria más lenta. La estrategia a seguir consiste en organizar los datos y los programas en memoria de manera que las palabras de memoria necesarias estén normalmente en la memoria más rápida.
- ➤ En general, es probable que la mayoría de los accesos futuros a la memoria principal, por parte del procesador, sean a posiciones accedidas recientemente. Por eso la caché automáticamente retiene una copia de algunas de las palabras de la DRAM utilizadas recientemente. Si la caché se diseña adecuadamente, la mayor parte del tiempo el procesador solicitará palabras de memoria que están ya en la caché.

Las memorias de los computadores, aunque parezcan conceptualmente sencillas, presentan tal vez la más amplia diversidad de tipos, tecnología, estructura, prestaciones y coste, de entre todos los componentes de un computador. Ninguna tecnología es óptima para satisfacer las necesidades de memoria de un computador. En consecuencia, un computador convencional está equipado con una jerarquía de subsistemas de memoria, algunos internos (directamente accesibles por el procesador), y otros externos (accesibles por el procesador mediante módulos de entrada/salida).

Este capítulo y el siguiente se centran en el estudio de la memoria interna, mientras que el Capítulo 6 se dedicará a la memoria externa. Para comenzar, en la primera sección de este capítulo examinaremos características clave de las memorias de un computador. El resto del capítulo se dedica al estudio de un elemento esencial de cualquier computador moderno: la memoria caché.

4.1. CONCEPTOS BÁSICOS SOBRE SISTEMAS DE MEMORIA DE COMPUTADORES

CARACTERÍSTICAS DE LOS SISTEMAS DE MEMORIA

El complejo tema de las memorias es más abordable si clasificamos los sistemas de memoria según sus características clave. Las más importantes se listan en la Tabla 4.1.

Tabla 4.1. Características clave de los sistemas de memoria de computadores.

Ubicación Procesador Interna (principal) Externa (secundaria)	**Pestaciones** Tiempo de acceso Tiempo de ciclo Velocidad de transferencia
Capacidad Tamaño de la palabra Número de palabras	**Dispositivo físico** Semiconductor Soporte magnético Soporte óptico Magneto-óptico
Unidad de transferencia Palabra Bloque	**Características físicas** Volátil/no volátil Borrable/no borrable
Método de acceso Acceso secuencial Acceso directo Acceso aleatorio Acceso asociativo	**Organización**

El término **ubicación** que aparece en la Tabla 4.1 indica si la memoria es interna o externa al computador. La memoria interna suele identificarse con la memoria principal. Sin embargo hay además otras formas de memoria interna. El procesador necesita su propia memoria local en forma de registros (*véase* por ejemplo la Figura 2.3). Además, como veremos, la unidad de control del procesador también puede necesitar su propia memoria interna. Postponemos la discusión de estos dos últimos tipos de memoria interna para capítulos posteriores. La memoria caché es también otro tipo de memoria interna. La memoria externa consta de dispositivos periféricos de almacenamiento, tales como discos y cintas, que son accesibles por el procesador a través de controladores de E/S.

Una característica obvia de las memorias es su **capacidad**. Para memorias internas se expresa normalmente en términos de bytes (1 byte = 8 bits) o de palabras. Longitudes de palabra comunes son 8, 16, y 32 bits. La capacidad de las memorias externas se suele expresar en bytes.

Un concepto relacionado es la **unidad de trasferencia**. Para memorias internas, la unidad de transferencia es igual al número de líneas de entrada/salida de datos del módulo de memoria. A menudo es igual a la longitud de palabra, pero suele ser mayor, por ejemplo 64, 128, o 256 bits. Para aclararlo consideremos tres conceptos relacionados con la memoria interna:

- **Palabra:** es la unidad «natural» de organización de la memoria. El tamaño de la palabra suele coincidir con el número de bits utilizados para representar números y con la longitud de las instrucciones. Por desgracia hay muchas excepciones. Por ejemplo, el CRAY C90 tiene una longitud de palabra de 64 bits, pero utiliza una representación de números enteros de 46 bits. El VAX tiene una gran variedad de longitudes de instrucción, expresadas como múltiplos de bytes, y una longitud de palabra de 32 bits.
- **Unidades direccionables:** en algunos sistemas la unidad direccionable es la palabra. Sin embargo muchos de ellos permiten direccionar a nivel de bytes. En cualquier caso, la relación entre la longitud *A* de una dirección y el número *N* de unidades direccionables, es $2^A = N$.

- **Unidad de transferencia:** para la memoria principal es el número de bits que se leen o escriben en memoria a la vez. La unidad de transferencia no tiene por qué coincidir con una palabra o con una unidad direccionable. Para la memoria externa, los datos se transfieren normalmente en unidades más grandes que la palabra denominadas bloques.

Otro distintivo entre tipos de memorias es el **método de acceso**, que incluye las siguientes variantes:

- **Acceso secuencial:** la memoria se organiza en unidades de datos llamadas registros. El acceso debe realizarse con una secuencia lineal específica. Se hace uso de información almacenada de direccionamiento que permite separar los registros y ayudar en el proceso de recuperación de datos. Se utiliza un mecanismo de lectura/escritura compartida que debe ir trasladándose desde su posición actual a la deseada, pasando y obviando cada registro intermedio. Así pues, el tiempo necesario para acceder a un registro dado es muy variable. Las unidades de cinta que se tratan en el Capítulo 6 son de acceso secuencial.
- **Acceso directo:** como en el caso de acceso secuencial, el directo tiene asociado un mecanismo de lectura/escritura. Sin embargo, los bloques individuales o registros tienen una dirección única basada en su dirección física. El acceso se lleva a cabo mediante un acceso directo a una vecindad dada, seguido de una búsqueda secuencial, bien contando, o bien esperando hasta alcanzar la posición final. De nuevo el tiempo de acceso es variable. Las unidades de disco, que se tratan en el Capítulo 6, son de acceso directo.
- **Acceso aleatorio (*random*):** cada posición direccionable de memoria tiene un único mecanismo de acceso cableado físicamente. El tiempo para acceder a una posición dada es constante e independiente de la secuencia de accesos previos. Por tanto, cualquier posición puede seleccionarse «aleatoriamente» y ser direccionada y accedida directamente. La memoria principal y algunos sistemas de caché son de acceso aleatorio.
- **Asociativa:** es una memoria del tipo de acceso aleatorio que permite hacer una comparación de ciertas posiciones de bits dentro de una palabra buscando que coincidan con unos valores dados, y hacer esto para todas las palabras simultáneamente. Una palabra es por tanto recuperada basándose en una porción de su contenido en lugar de su dirección. Como en las memorias de acceso aleatorio convencionales, cada posición tiene su propio mecanismo de direccionamiento, y el tiempo de recuperación de un dato es una constante independiente de la posición o de los patrones de acceso anteriores. Las memorias caché pueden emplear acceso asociativo.

Desde el punto de vista del usuario, las dos características más importantes de una memoria son su capacidad y sus **prestaciones**. Se utilizan tres parámetros de medida de prestaciones:

- **Tiempo de acceso (latencia):** para memorias de acceso aleatorio es el tiempo que tarda en realizarse una operación de escritura o de lectura, es decir, el tiempo que transcurre desde el instante en el que se presenta una dirección a la memoria hasta que el dato, o ha sido memorizado, o está disponible para su uso. Para memorias de otro tipo, el tiempo de acceso es el que se tarda en situar el mecanismo de lectura/escritura en la posición deseada.
- **Tiempo de ciclo de memoria:** este concepto se aplica principalmente a las memorias de acceso aleatorio y consiste en el tiempo de acceso y algún tiempo más que se requiere antes de que pueda iniciarse un segundo acceso a memoria. Este tiempo adicional puede que sea necesario

para que finalicen las transiciones en las líneas de señal o para regenerar los datos en el caso de lecturas destructivas. Tenga en cuenta que el tiempo de ciclo de memoria depende de las características del bus del sistema y no del procesador.

- **Velocidad de transferencia:** es la velocidad a la que se pueden transferir datos a, o desde, una unidad de memoria. Para memorias de acceso aleatorio coincide con el inverso del tiempo de ciclo.

Para otras memorias se utiliza la siguiente relación:

$$T_N = T_A + \frac{N}{R}$$

donde:

T_N = Tiempo medio de escritura o de lectura de N bits

T_A = Tiempo de acceso medio

N = Número de bits

R = Velocidad de transferencia, en bits por segundo (bps)

Se han empleado **soportes físicos** muy diversos para las memorias. Las más comunes en la actualidad son las memorias semiconductoras, las memorias de superficie magnética, utilizadas para discos y cintas, y las memorias ópticas y magneto-ópticas.

Del almacenamiento de datos son importantes varias **características físicas**. En memorias volátiles la información se va perdiendo o desaparece cuando se desconecta la alimentación. En las memorias no volátiles la información, una vez grabada, permanece sin deteriorarse hasta que se modifique intencionadamente; no se necesita la fuente de alimentación para retener la información. Las memorias de superficie magnética son no volátiles. Las memorias semiconductoras pueden ser volátiles o no volátiles. Las memorias no borrables no pueden modificarse, salvo que se destruya la unidad de almacenamiento. Las memorias semiconductoras de este tipo se conocen por el nombre de *memorias de solo lectura* (ROM, *Read Only Memory*). Una memoria no borrable es necesariamente no volátil.

En memorias de acceso aleatorio, su **organización** es un aspecto clave de diseño. Por *organización* se entiende su disposición o estructura física en bits para formar palabras. Como explicaremos pronto, la estructura más obvia no es siempre la utilizada en la práctica.

JERARQUÍA DE MEMORIA

Las restricciones de diseño de la memoria de un computador se pueden resumir en tres cuestiones: ¿cuánta capacidad? ¿cómo de rápida? ¿de qué coste?

La cuestión del tamaño es un tema siempre abierto. Si se consigue hasta una cierta capacidad, probablemente se desarrollarán aplicaciones que la utilicen. La cuestión de la rapidez es, en cierto sentido, fácil de responder. Para conseguir las prestaciones óptimas, la memoria debe seguir al procesador. Es decir, cuando el procesador ejecuta instrucciones, no es deseable que tenga que detenerse a la espera de instrucciones o de operandos. La última de las cuestiones anteriores también debe tenerse en cuenta. En la práctica, el coste de la memoria debe ser razonable con relación a los otros componentes.

Como es de esperar, existe un compromiso entre las tres características clave de coste, capacidad, y tiempo de acceso. En un momento dado, se emplean diversas tecnologías para realizar los sistemas de memoria. En todo el espectro de posibles tecnologías se cumplen las siguientes relaciones:

- A menor tiempo de acceso, mayor coste por bit.
- A mayor capacidad, menor coste por bit.
- A mayor capacidad, mayor tiempo de acceso.

El dilema con que se enfrenta el diseñador está claro. El diseñador desearía utilizar tecnologías de memoria que proporcionen gran capacidad, tanto porque esta es necesaria como porque el coste por bit es bajo. Sin embargo, para satisfacer las prestaciones requeridas, el diseñador necesita utilizar memorias costosas, de capacidad relativamente baja y con tiempos de acceso reducidos.

La respuesta a este dilema es no contar con un solo componente de memoria, sino emplear una **jerarquía de memoria**. La Figura 4.1 ilustra una jerarquía típica. Cuando se desciende en la jerarquía ocurre:

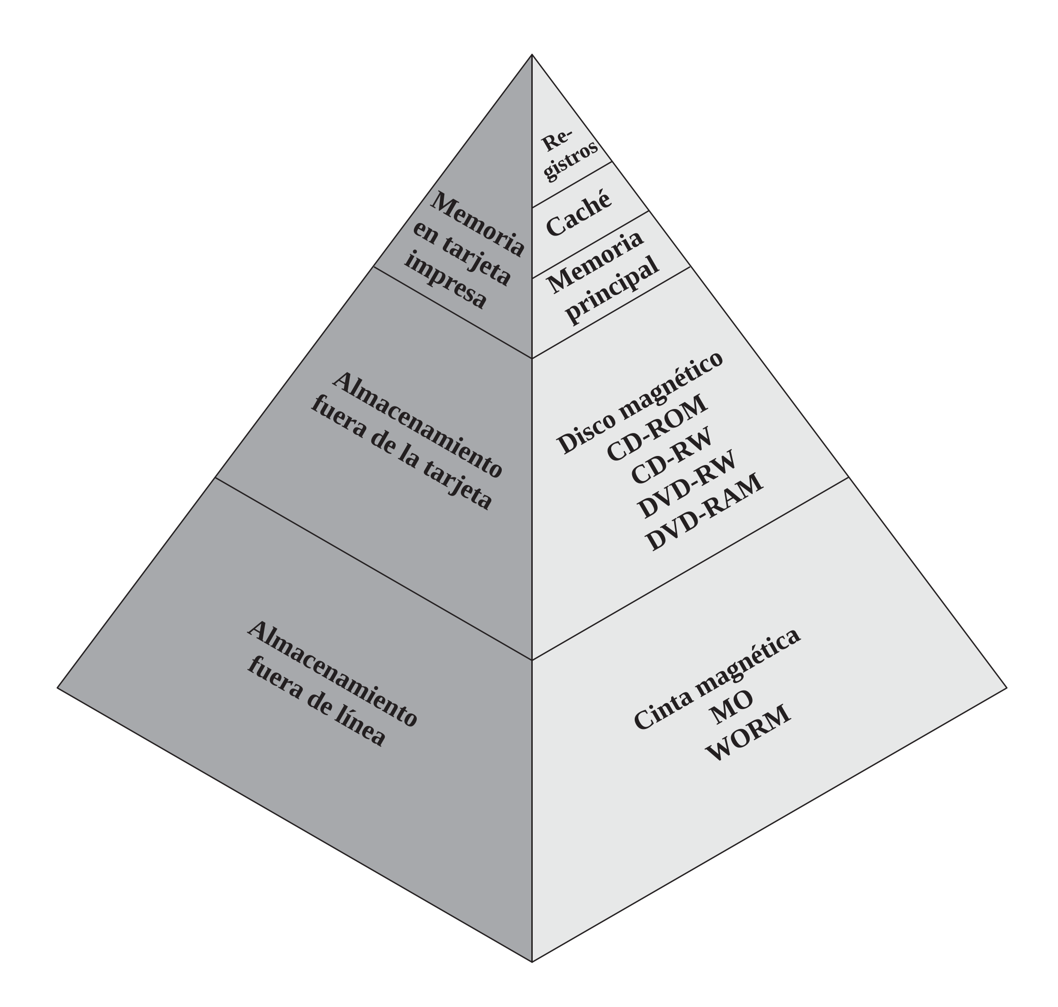


Figura 4.1. Jerarquía de memoria.

a) Disminuye el coste por bit.

b) Aumenta la capacidad.

c) Aumenta el tiempo de acceso.

d) Disminuye la frecuencia de accesos a la memoria por parte del procesador.

Así pues, memorias más pequeñas, más costosas y más rápidas, se complementan con otras más grandes, más económicas y más lentas. La clave del éxito de esta organización está en el último *item* (d): la disminución de la frecuencia de acceso. Examinaremos con detalle este concepto cuando hablemos de la caché (después, en este mismo capítulo) y de la memoria virtual (en el Capítulo 8), pero daremos aquí una breve explicación.

Ejemplo 4.1. Supongamos que el procesador tiene que acceder a dos niveles de la memoria. El nivel 1 contiene 1 000 palabras y tiene un tiempo de acceso de 0,01 μs. El nivel 2 contiene 100 000 palabras y tiene un tiempo de acceso de 0,1 μs. Supongamos que si la palabra a la que se va a acceder está en el nivel 1, el procesador accede a ella directamente. Si está en el nivel 2, entonces es primeramente transferida al nivel 1 y después accedida por el procesador. Por simplicidad ignoramos el tiempo necesario para que el procesador determine si la palabra está en un nivel u otro. La Figura 4.2 muestra la forma que en general tiene la curva que representa esta situación. La figura muestra el tiempo de acceso medio a una memoria de dos niveles, en función de la tasa de acierto H, donde H se define como la fracción del total de accesos a memoria encontrados en la memoria más rápida (por ejemplo, en la caché); T_1 es el tiempo de acceso al nivel 1, y T_2 el tiempo de acceso al nivel 2[1]. Como puede verse, para porcentajes altos de accesos al nivel 1, el tiempo de acceso total promedio es mucho más próximo al del nivel 1 que al del nivel 2.

En nuestro ejemplo, si suponemos que el 95 por ciento de los accesos a memoria se encuentran con éxito en la caché, entonces el tiempo medio para acceder a una palabra puede expresarse en la forma:

$$(0{,}95)\ (0{,}01\ \mu s) + (0{,}05)\ (0{,}01\ \mu s + 0{,}1\ \mu s) = 0{,}0095\ \mu s + 0{,}0055\ \mu s = 0{,}015\ \mu s$$

Como era deseable, el tiempo de acceso medio está mucho más próximo a 0,01 μs que a 0,1 μs.

En principio, el uso de dos niveles de memoria para reducir el tiempo de acceso medio funciona, pero solo si se aplican las condiciones (a) a (d) anteriores. Empleando diversas tecnologías se tiene todo un espectro de sistemas de memoria que satisfacen las condiciones (a) a (c). Afortunadamente, la condición (d) es también generalmente valida.

La base para la validez de la condición (d) es el principio conocido como **localidad de las referencias** [DENN68]. En el curso de la ejecución de un programa, las referencias a memoria por parte del procesador, tanto para instrucciones como para datos, tienden a estar agrupadas. Los programas normalmente contienen un número de bucles iterativos y subrutinas. Cada vez que se entra en un bucle o una subrutina, hay repetidas referencias a un pequeño conjunto de instrucciones. De manera similar, las

[1] Si la palabra accedida se encontraba en la memoria más rápida, se dice que se ha producido un **acierto**. Y si no se encontraba en la memoria más rápida, se dice que ha tenido lugar un **fallo.**

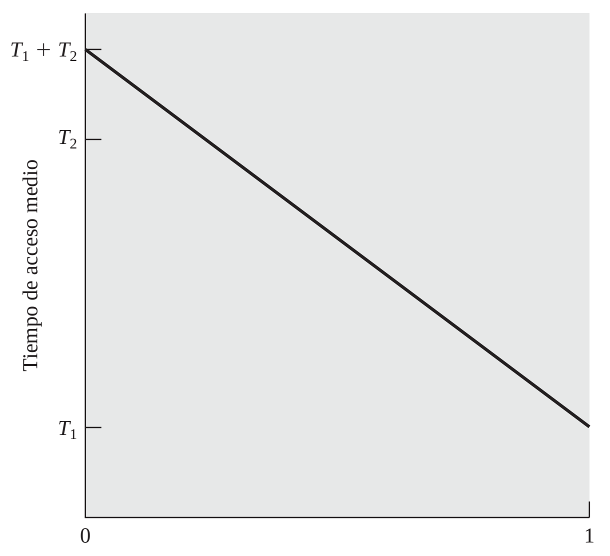


Figura 4.2. Prestaciones de una memoria de dos niveles sencilla.

operaciones con tablas o con matrices conllevan accesos a un conjunto de palabras de datos agrupadas. En periodos de tiempo largos, las agrupaciones (*clusters*) en uso cambian, pero en periodos de tiempo cortos, el procesador trabaja principalmente con agrupaciones fijas de referencias a memoria.

De acuerdo con lo anterior, es posible organizar los datos a través de la jerarquía de tal manera que el porcentaje de accesos a cada nivel siguiente más bajo sea sustancialmente menor que al nivel anterior. Considérese el ejemplo de dos niveles ya presentado, y que la memoria del nivel 2 contiene todos los datos e instrucciones de programa. Las agrupaciones actuales pueden ubicarse temporalmente en el nivel 1. De vez en cuando, una de las agrupaciones del nivel 1 tendrá que ser devuelta al nivel 2 a fin de que deje sitio para que entre otra nueva agrupación al nivel 1. En general, sin embargo, la mayoría de las referencias serán a instrucciones y datos contenidos en el nivel 1.

Este principio puede aplicarse a través de más de dos niveles de memoria, como sugiere la jerarquía mostrada en la Figura 4.1. El tipo de memoria más rápida, pequeña y costosa, lo constituyen los registros internos al procesador. Un procesador suele contener unas cuantas docenas de tales registros, aunque algunas máquinas contienen cientos de ellos. Descendiendo dos niveles, la memoria principal es el principal sistema de memoria interna del computador. Cada posición de memoria principal tiene una única dirección. La memoria principal es normalmente ampliada con una caché, que es más pequeña y rápida. La caché no suele estar visible al programador, y realmente tampoco al procesador. Es un dispositivo para escalonar las transferencias de datos entre memoria principal y los registros del procesador a fin de mejorar las prestaciones.

Las tres formas de memoria que acabamos de describir son, normalmente, volátiles y de tecnología semiconductora. El uso de tres niveles aprovecha la variedad existente de tipos de memorias semiconductoras, que difieren en velocidad y coste. El almacenamiento de datos de forma más permanente se hace en dispositivos de memoria masiva, de los cuales los más comunes son el disco duro y los dispositivos extraíbles, tales como discos extraíbles, cintas y dispositivos ópticos de almacenamiento.

Las memorias externas no volátiles o permanentes se denominan también memorias secundarias o auxiliares. Se utilizan para almacenar programas y ficheros de datos, y suelen estar visibles al programador solo en términos de ficheros y registros, en lugar de bytes aislados o de palabras. El disco se emplea además para proporcionar una ampliación de la memoria principal conocida como memoria virtual, que será tratada en el Capítulo 8.

En la jerarquía pueden incluirse otras formas de memoria. Por ejemplo, los grandes computadores de IBM incluyen una forma de memoria interna conocida como almacenamiento extendido. Este utiliza una tecnología semiconductora que es más lenta y menos costosa que la de la memoria principal. Estrictamente hablando, esta memoria no encaja en la jerarquía sino que es una ramificación lateral: los datos pueden transferirse entre memoria principal y el almacenamiento extendido pero no entre este y la memoria externa. Otras formas de memoria secundaria incluyen los discos ópticos y los magneto-ópticos. Finalmente, mediante software se pueden añadir más niveles a la jerarquía. Una parte de la memoria principal puede utilizarse como almacén intermedio (*buffer*) para guardar temporalmente datos que van a ser volcados en disco. Esta técnica, a veces denominada caché de disco [2], mejora las prestaciones de dos maneras:

- Las escrituras en disco se hacen por grupos. En lugar de muchas transferencias cortas de datos, tenemos pocas transferencias largas. Esto mejora las prestaciones del disco y minimiza la participación del procesador.
- Algunos datos destinados a ser escritos como salidas pueden ser referenciados por un programa antes de que sean volcados en disco. En ese caso, los datos se recuperan rápidamente desde la caché software en lugar de hacerlo lentamente de disco.

El Apéndice 4A examina las implicaciones sobre prestaciones de las estructuras de memoria multinivel.

4.2. PRINCIPIOS BÁSICOS DE LAS MEMORIAS CACHÉ

El objetivo de la memoria caché es lograr que la velocidad de la memoria sea lo más rápida posible, consiguiendo al mismo tiempo un tamaño grande al precio de memorias semiconductoras menos costosas. El concepto se ilustra en la Figura 4.3. Hay una memoria principal relativamente grande y más

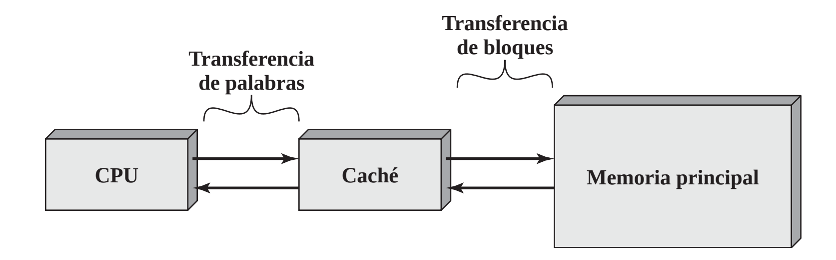


Figura 4.3. Memorias caché y principal.

[2] La caché de disco generalmente es una técnica software y no es estudiada en este libro. *Véase* [STAL05] para una discusión del tema.

lenta, junto con una memoria caché más pequeña y rápida. La caché contiene una copia de partes de la memoria principal. Cuando el procesador intenta leer una palabra de memoria, se hace una comprobación para determinar si la palabra está en la caché. Si es así, se entrega dicha palabra al procesador. Si no, un bloque de memoria principal, consistente en un cierto número de palabras, se transfiere a la caché y después la palabra es entregada al procesador. Debido al fenómeno de localidad de las referencias, cuando un bloque de datos es capturado por la caché para satisfacer una referencia a memoria simple, es probable que se hagan referencias futuras a la misma posición de memoria o a otras palabras del mismo bloque.

La Figura 4.4 describe la estructura de un sistema de memoria caché/principal. La memoria principal consta de hasta 2^n palabra direccionables, teniendo cada palabra una única dirección de n bits. Esta memoria la consideramos dividida en un número de bloques de longitud fija, de K palabras por bloque. Es decir, hay $M = 2^n/K$ bloques. La caché consta de C **líneas.** Cada línea contiene K palabras, más una etiqueta de unos cuantos bits; denominándose tamaño de línea al número de palabras que hay en la línea. El número de líneas es considerablemente menor que el número de bloques de memoria principal ($C << M$). En todo momento, un subconjunto de los bloques de memoria reside en líneas de la caché. Si se lee una palabra de un bloque de memoria, dicho bloque es transferido a una de las líneas de la caché. Ya que hay más bloques que líneas, una línea dada no puede dedicarse unívoca y permanentemente a un bloque. Por consiguiente, cada línea incluye una **etiqueta** que identifica qué bloque particular almacena. La etiqueta es usualmente una porción de la dirección de memoria principal, como describiremos más adelante en esta sección.

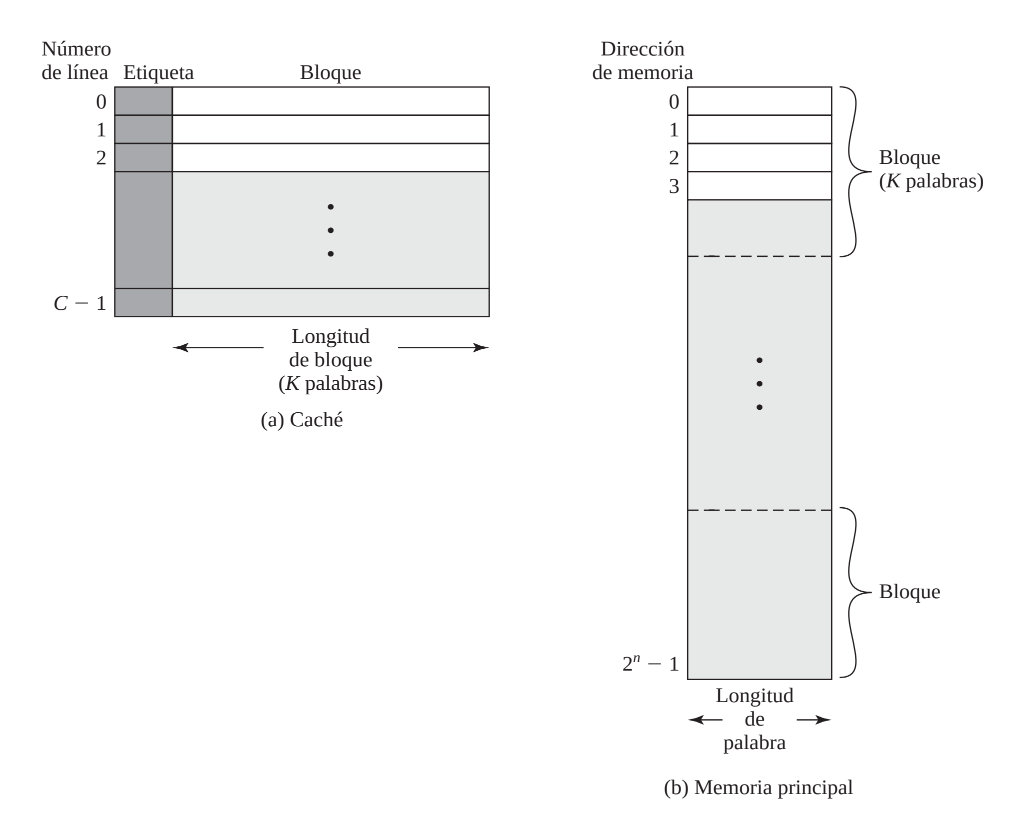


Figura 4.4. Estructura de memoria caché/principal.

La Figura 4.5 ilustra una operación de lectura. El procesador genera la dirección, RA, de una palabra a leer. Si la palabra está en la caché, es entregada al procesador. Si no, el bloque que contiene dicha palabra se carga en la caché, y la palabra después es llevada al procesador. La Figura 4.5 indica cómo estas dos últimas operaciones se realizan en paralelo y refleja la organización mostrada en la Figura 4.6, que es típica en las organizaciones de caché actuales. En ella, la caché conecta con el procesador mediante líneas de datos, de control y de direcciones. Las líneas de datos y de direcciones conectan también con buffers de datos y de direcciones que las comunican con un bus del sistema a través del cual se accede a la memoria principal. Cuando ocurre un acierto de caché, los buffers de datos y de direcciones se inhabilitan, y la comunicación tiene lugar solo entre procesador y caché, sin tráfico en el bus. Cuando ocurre un fallo de caché, la dirección deseada se carga en el bus del sistema y el dato es llevado, a través del buffer de datos, tanto a la caché como al procesador. En otras formas de organización, la caché se interpone físicamente entre el procesador y la memoria

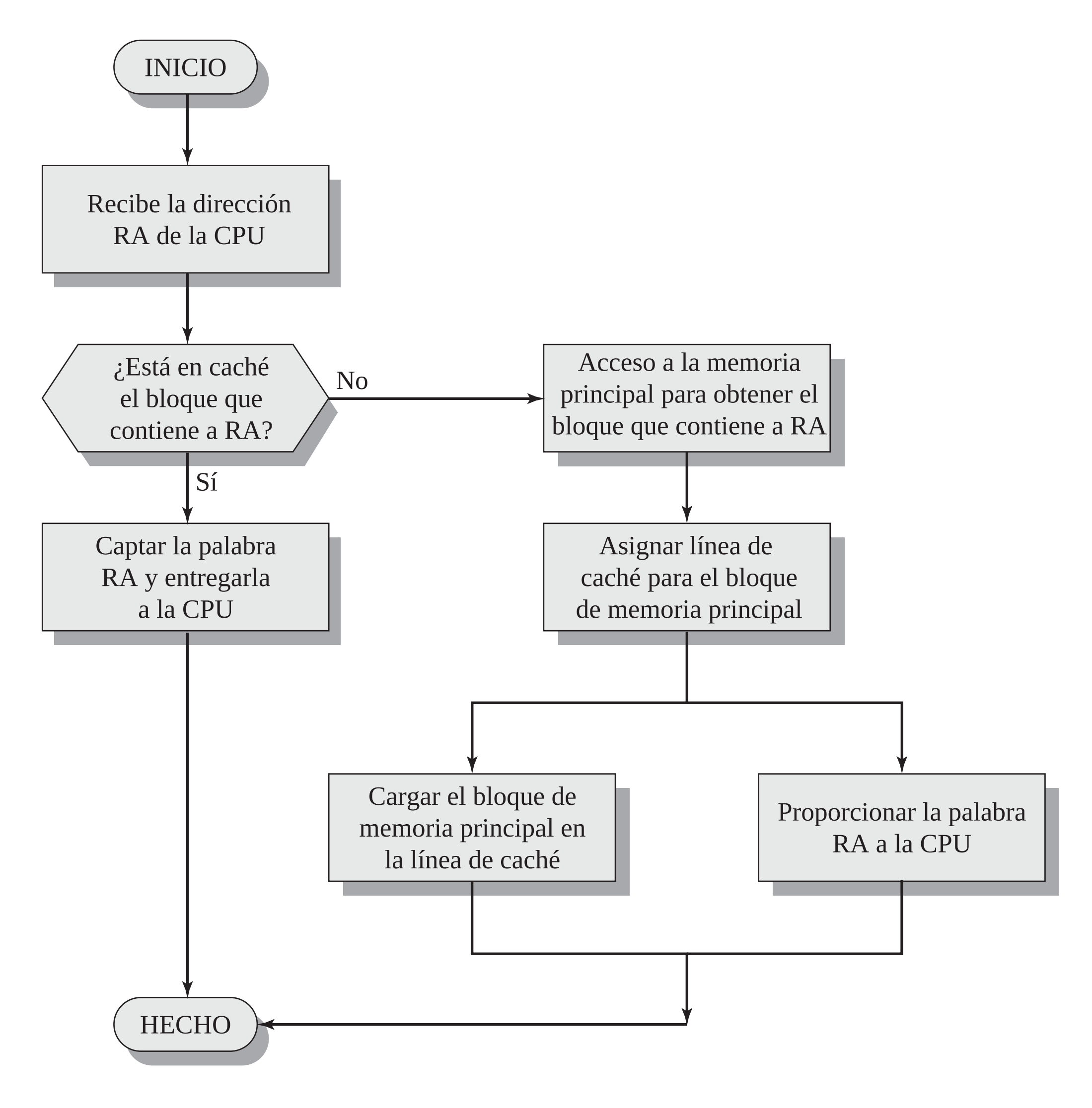


Figura 4.5. Operación de lectura de caché.

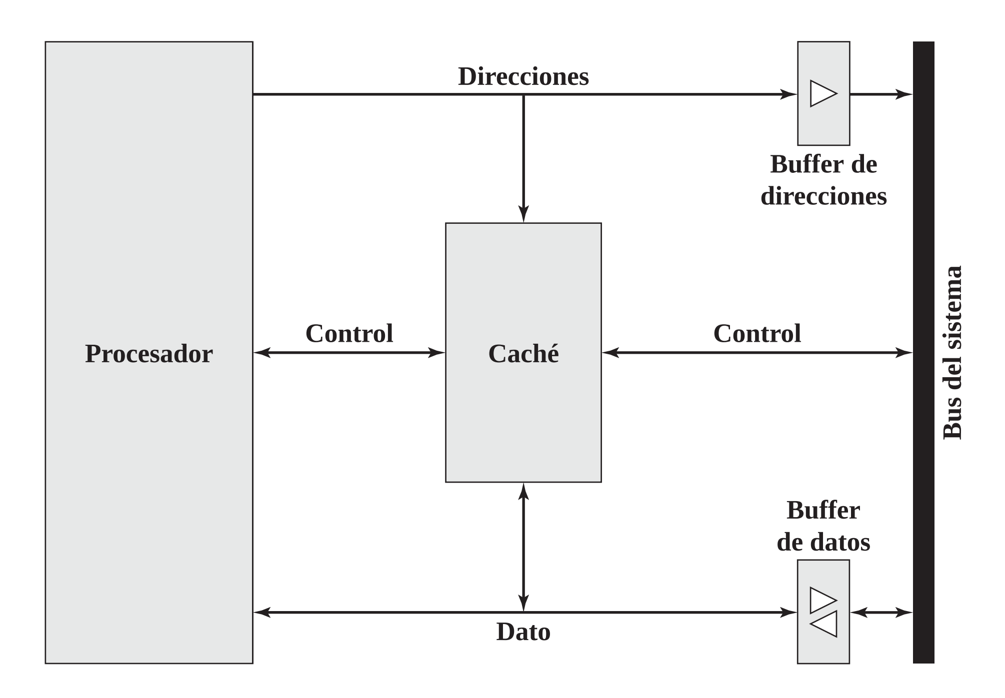


Figura 4.6. Organización típica de caché.

principal para todas las líneas de datos, direcciones y control. En este caso, frente a un fallo de caché, la palabra deseada es primero leída por la caché y después transferida desde esta al procesador.

El Apéndice 4A contiene un análisis de los parámetros de prestaciones relativos al uso de la caché.

4.3. ELEMENTOS DE DISEÑO DE LA CACHÉ

En esta sección se revisan los parámetros de diseño de la caché y se muestran algunos resultados típicos. A veces nos referimos al uso de cachés en el contexto de la computación de altas prestaciones (HPC, *High Performancea Computing*). La HPC trata los supercomputadores y su programación, especialmente para aplicaciones científicas que implican grandes cantidades de datos, cálculos con vectores y matrices, y el uso de algoritmos paralelos. El diseño de cachés para HPC difiere bastante del diseño para otras plataformas hardware y aplicaciones. Realmente, diversos investigadores han concluido que las aplicaciones de HPC presentan unas prestaciones pobres en arquitecturas de computadores que emplean cachés [BAIL93]. Desde entonces, otros investigadores han mostrado que una jerarquía de cachés puede ser útil para mejorar las prestaciones si el software de aplicación permite una explotación adecuada de la caché [WANG99, PRES01] [3].

Aunque hay muy diversas implementaciones de caché, existen unos cuantos criterios básicos de diseño que sirven para clasificar y diferenciar entre arquitecturas de caché. La Tabla 4.2 lista algunos elementos clave.

---

[3] *Véase* [DOWD98] para un tratamiento más general de la HPC.

Tabla 4.2. Elementos de diseño de la caché.

Tamaño de caché	**Política de escritura**
Función de correspondencia	Escritura inmediata
Directa	Postescritura
Asociativa	Escritura única
Asociativa por conjuntos	**Tamaño de línea**
Algoritmo de sustitución	**Número de cachés**
Utilizado menos recientemente (LRU)	Uno o dos niveles
Primero en entrar-primero en salir (FIFO)	Unificada o partida
Utilizado menos frecuentemente (LFU)	
Aleatorio	

TAMAÑO DE CACHÉ

El primer elemento, el tamaño de caché, ya ha sido tratado. Nos gustaría que el tamaño fuera lo suficientemente pequeño como para que el coste total medio por bit se aproxime al de la memoria principal sola, y que fuera lo suficientemente grande como para que el tiempo de acceso medio total sea próximo al de la caché sola. Hay otras muchas motivaciones para minimizar el tamaño de la caché. Cuanto más grande es, mayor es el número de puertas implicadas en direccionar la caché. El resultado es que cachés grandes tienden a ser ligeramente más lentas que las pequeñas (incluso estando fabricadas con la misma tecnología de circuito integrado y con la misma ubicación en el chip o en la tarjeta de circuito impreso). El tamaño de caché está también limitado por las superficies disponibles de chip y de tarjeta. Como las prestaciones de la caché son muy sensibles al tipo de tarea, es imposible predecir un tamaño «óptimo». La Tabla 4.3 lista los tamaños de caché de diversos procesadores antiguos y modernos.

FUNCIÓN DE CORRESPONDENCIA

Ya que hay menos líneas de caché que bloques de memoria principal, se necesita un algoritmo que haga corresponder bloques de memoria principal a líneas de caché. Además, se requiere algún medio para determinar qué bloque de memoria principal ocupa actualmente una línea dada de caché. La elección de la función de correspondencia determina cómo se organiza la caché. Pueden utilizarse tres técnicas: directa, asociativa, y asociativa por conjuntos. Examinamos a continuación cada una de ellas. En cada caso veremos la estructura general y un ejemplo concreto.

Ejemplo 4.2. Para los tres casos, el ejemplo incluye los siguientes elementos:

- La caché puede almacenar 64 KB.
- Los datos se transfieren entre la memoria principal y la caché en bloques de 4 bytes. Esto significa que la caché está organizada en 16K = 2^{14} líneas de 4 bytes cada una.
- La memoria principal es de 16MB, con cada byte directamente direccionable mediante una dirección de 24 bits (2^{24} = 16M). Así pues, al objeto de realizar la correspondencia, podemos considerar que la memoria principal consta de 4M bloques de 4 bytes cada uno.

Tabla 4.3. Tamaños de caché de algunos procesadores.

Procesador	Tipo	Año de introducción	Caché L1[a]	Caché L2	Caché L3
IBM 360/55	Gran computador	1968	16 a 32 KB	—	—
PDP-11/70	Minicomputador	1975	1 KB	—	—
VAX 11/780	Minicomputador	1978	16 KB	—	—
IBM 3033	Gran computador	1978	64 KB	—	—
IBM 3090	Gran computador	1985	128 a 26 KB	—	—
Interl 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 a 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC 64	PC/servidor	1999	32 KB/32 KB	256 KM 1 MB	2 MB
IBM S/390 G4	Gran computador	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Gran computador	1999	256 KB	8 MB	—
Pentium 4	PC/servidor	2000	8 KB/8 KB	256 KB	—
IBM SP	Servidor de gama alta/Supercomputador	2000	64 KB/32 KB	8 MB	—
CRAY MTA[b]	Supercomputador	2000	8 KB	2 MB	—
Itanium	PC/Servidor	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	Servidor de gama alta	2001	32 KB/32 KB	4 MB	—
Itanium 2	PC/Servidor	2002	32 KB	256 KB	6 MB
IBM POWER5	Servidor de gama alta	2003	64 KB	1,9 MB	36 MB
CRAY XD-1	Supercomputador	2004	64 KB/64 KB	1 MB	—

[a] Dos valores separados por una barra inclinada (/) hacen referencia a las cachés de instrucciones y de datos.
[b] Ambas cachés son de instrucciones; no caché de datos.

Correspondencia directa. La técnica más sencilla, denominada correspondencia directa, consiste en hacer corresponder cada bloque de memoria principal a solo una línea posible de caché. La Figura 4.7 ilustra el mecanismo general. La correspondencia se expresa como:

$$i = j \text{ módulo } m$$

donde

i = número de línea de caché

j = número de bloque de memoria principal

m = número de líneas en la caché

La función de correspondencia se implementa fácilmente utilizando la dirección. Desde el punto de vista del acceso a caché, cada dirección de memoria principal puede verse como dividida en tres

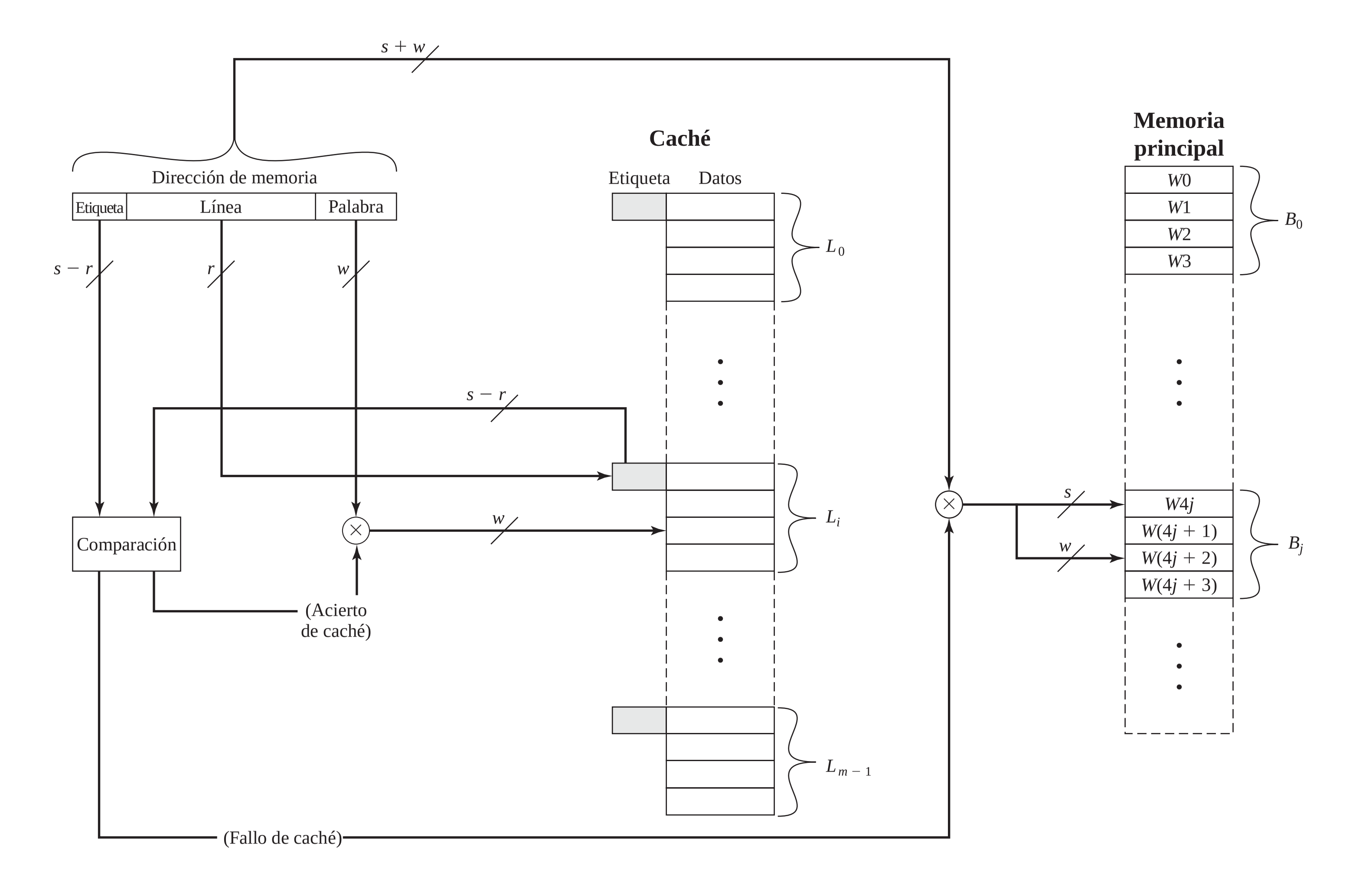


Figura 4.7. Organización de caché con correspondencia directa [HWAN93].

campos. Los w bits menos significativos identifican cada palabra dentro de un bloque de memoria principal; en la mayoría de las máquinas actuales, el direccionamiento es a nivel de bytes. Los s bits restantes especifican uno de los 2^s bloques de la memoria principal. La lógica de la caché interpreta estos s bits como una etiqueta de $s - r$ bits (parte más significativa) y un campo de línea de r bits. Este último campo identifica una de las $m = 2^r$ líneas de la caché.

Resumiendo:

- Longitud de las direcciones = $(s + w)$ bits
- Número de unidades direccionables = 2^{s+w} palabras o bytes
- Tamaño de bloque = tamaño de línea = 2^w palabras o bytes
- Número de bloques en memoria principal = $\frac{2^{s+w}}{2^w} = 2^s$
- Número de líneas en caché = $m = 2^r$
- Tamaño de la etiqueta = $(s - r)$ bits

El resultado es que se hacen corresponder bloques de memoria principal a líneas de caché de la siguiente manera:

Línea de caché	Bloques de memoria principal asignados
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m + 1, 2m + 1, \ldots, 2^s - m + 1$
.	.
$m - 1$	$m - 1, 2m - 1, 3m - 1, \ldots, 2^s - 1$

Por tanto, el uso de una parte de la dirección como número de línea proporciona una correspondencia o asignación única de cada bloque de memoria principal en la caché. Cuando un bloque es realmente escrito en la línea que tiene asignada, es necesario etiquetarlo para distinguirlo del resto de los bloques que pueden introducirse en dicha línea. Para ello se emplean los s-r bits más significativos.

Ejemplo 4.2a. La Figura 4.8 muestra nuestro ejemplo de sistema utilizando correspondencia directa.[4] En el ejemplo: $m = 16\text{K} = 2^{14}$, $i = j$ módulo 2^{14}. La asignación sería:

Línea de caché	Dirección de memoria de comienzo de bloque
0	000000, 010000, …, FF0000
1	000004, 010004, …, FF0004
⋮	⋮
$2^{14} - 1$	00FFFC, 01FFFC, …, FFFFFC

[4] En esta y en figuras posteriores, las direcciones y valores de memoria se expresan en notación hexadecimal. El Apéndice A contiene un resumen de los sistemas de numeración (decimal, binario, hexadecimal).

Obsérvese que no hay dos bloques que se asignen en la misma línea que tengan el mismo número de etiqueta. Así, los bloques 000000, 010000..., FF0000 tienen respectivamente los números de etiqueta 00, 01..., FF.

Refiriéndonos de nuevo a la Figura 4.5, una operación de lectura se lleva a cabo de la siguiente manera. Al sistema de caché se presenta una dirección de 24 bits. El número de línea, de 14 bits, se utiliza como índice para acceder a una línea particular dentro de la caché. Si el número de etiqueta, de 8 bits, coincide con el número de etiqueta almacenado actualmente en esa línea, el número de palabra de 2 bits se utiliza para seleccionar uno de los cuatro bytes de esa línea. Si no, el campo de 22 bits de etiqueta+línea se emplea para captar un bloque de memoria principal. La dirección real que se utiliza para la captación consta de los mencionados 22 bits concatenados con dos bits 0, y se captan 4 bytes a partir del comienzo del bloque.

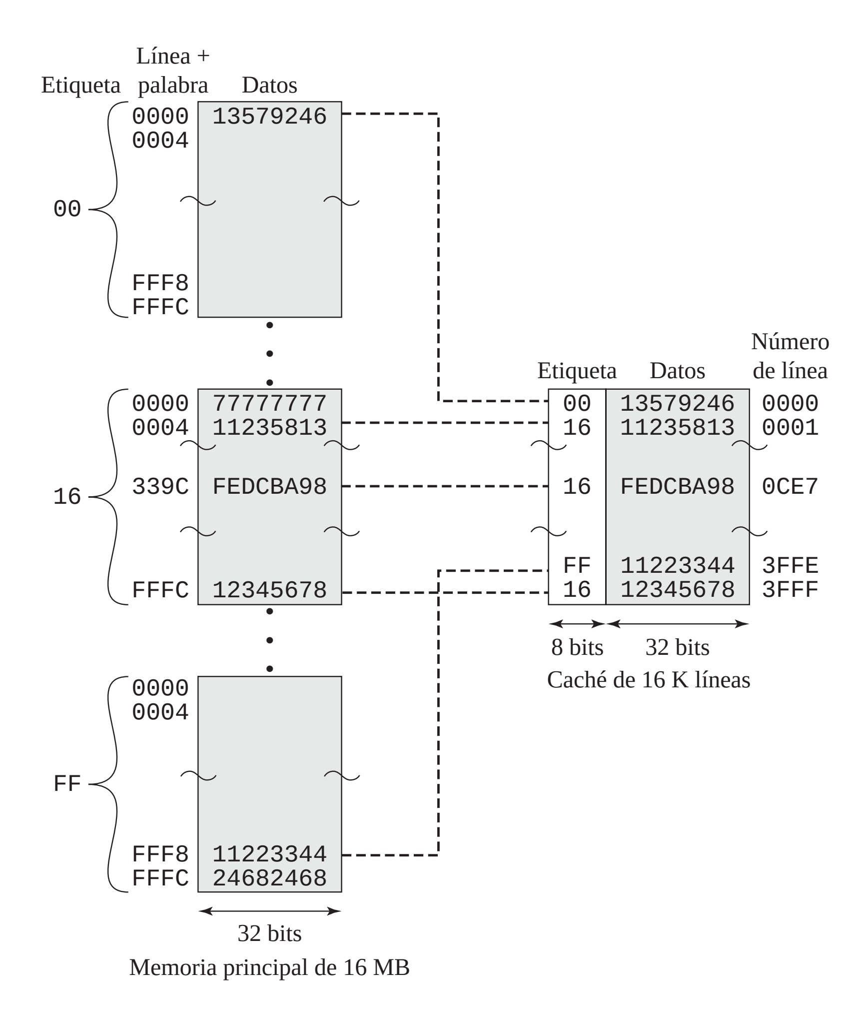


Figura 4.8. Ejemplo de correspondencia directa.

La técnica de correspondencia directa es sencilla y poco costosa de implementar. Su principal desventaja es que hay una posición concreta de caché para cada bloque dado. Por ello, si un programa referencia repetidas veces a palabras de dos bloques diferentes asignados en la misma línea, dichos bloques se estarían intercambiando continuamente en la caché, y la tasa de aciertos sería baja [un fenómeno conocido con el nombre de vapuleo (*thrashing*)].

Correspondencia asociativa. La correspondencia asociativa supera la desventaja de la directa, permitiendo que cada bloque de memoria principal pueda cargarse en cualquier línea de la caché. En este caso, la lógica de control de la caché interpreta una dirección de memoria simplemente como una etiqueta y un campo de palabra. El campo de etiqueta identifica unívocamente un bloque de memoria principal. Para determinar si un bloque está en la caché, su lógica de control debe examinar simultáneamente todas las etiquetas de líneas para buscar una coincidencia. La Figura 4.9 muestra esta lógica. Observe que ningún campo de la dirección corresponde al número de línea, de manera que el número de líneas de la caché no está fijado por el formato de las direcciones. En resumen:

- Longitud de las direcciones = $(s + w)$ bits
- Número de unidades direccionables = 2^{s+w} palabras o bytes
- Tamaño de bloque = tamaño de línea = 2^w palabras o bytes
- Número de bloques en memoria principal = $\frac{2^{s+w}}{2^w} = 2^s$
- Número de líneas en caché = indeterminado
- Tamaño de la etiqueta = s bits

Ejemplo 4.2b. La Figura 4.10 muestra nuestro ejemplo utilizando correspondencia asociativa. Una dirección de memoria principal consta de una etiqueta de 22 bits, más 2 bits que identifican un número de byte. La etiqueta de 22 bits debe almacenarse con el bloque de 32 bits de datos en cada línea de la caché. Obsérvese que son los 22 bits de la izquierda de la dirección (los más significativos) los que forman la etiqueta [5]. De manera que, la dirección de 24 bits 16339C en hexadecimal, contiene la etiqueta de 22 bits 058CE7. Esto se ve fácilmente en notación binaria:

dirección de memoria	0001	0110	0011	0011	1001	1100	(binario)
	1	6	3	3	9	C	(hexadecimal)
etiqueta	00	0101	1000	1100	1110	0111	(binario)
(22 bits de la izda.)	0	5	8	C	E	7	(hexadecimal)

[5] En la Figura 4.10, la etiqueta de 22 bits se representa mediante un número hexadecimal de seis dígitos; de los cuales el dígito más significativo tiene una longitud efectiva de solo dos bits.

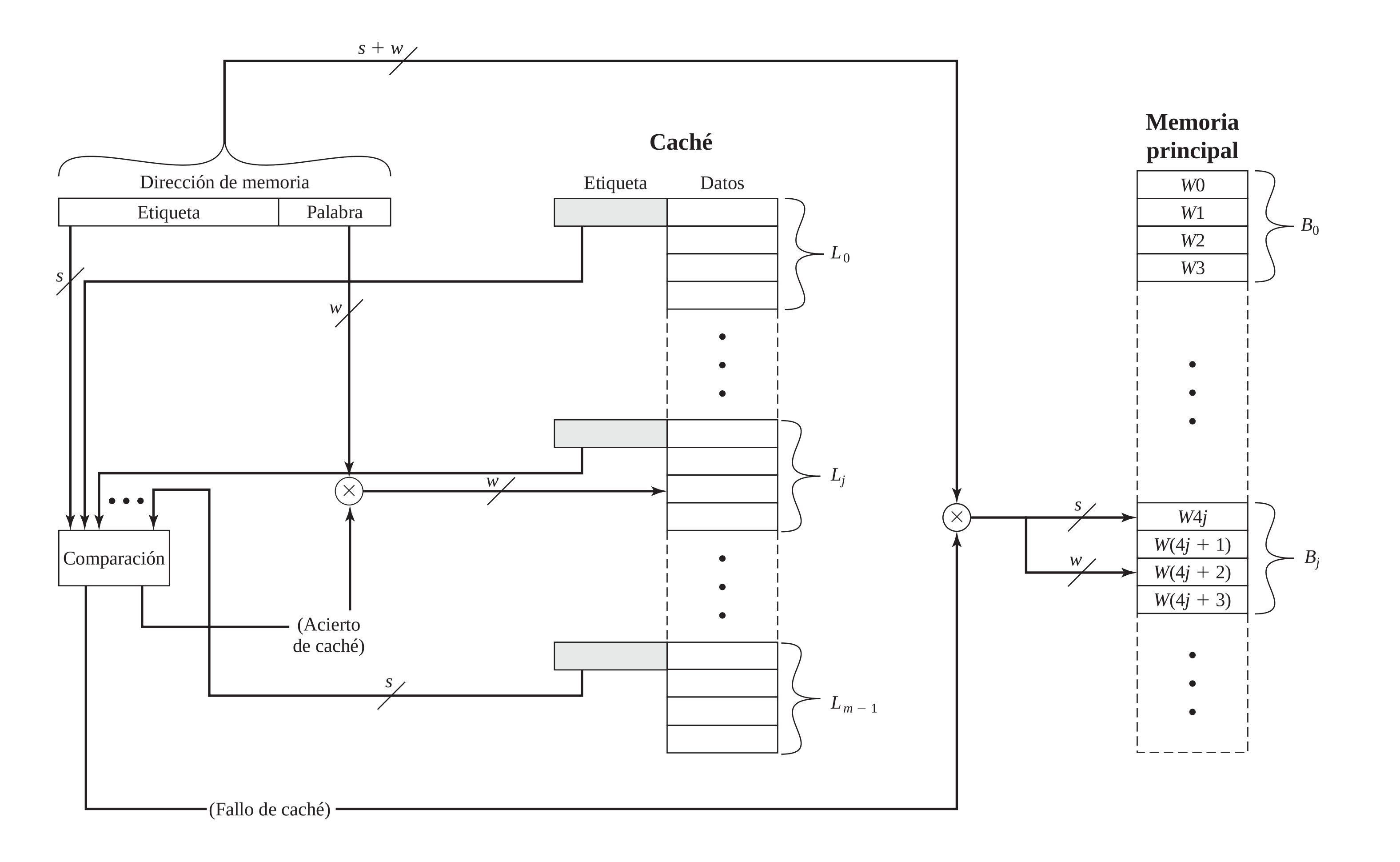
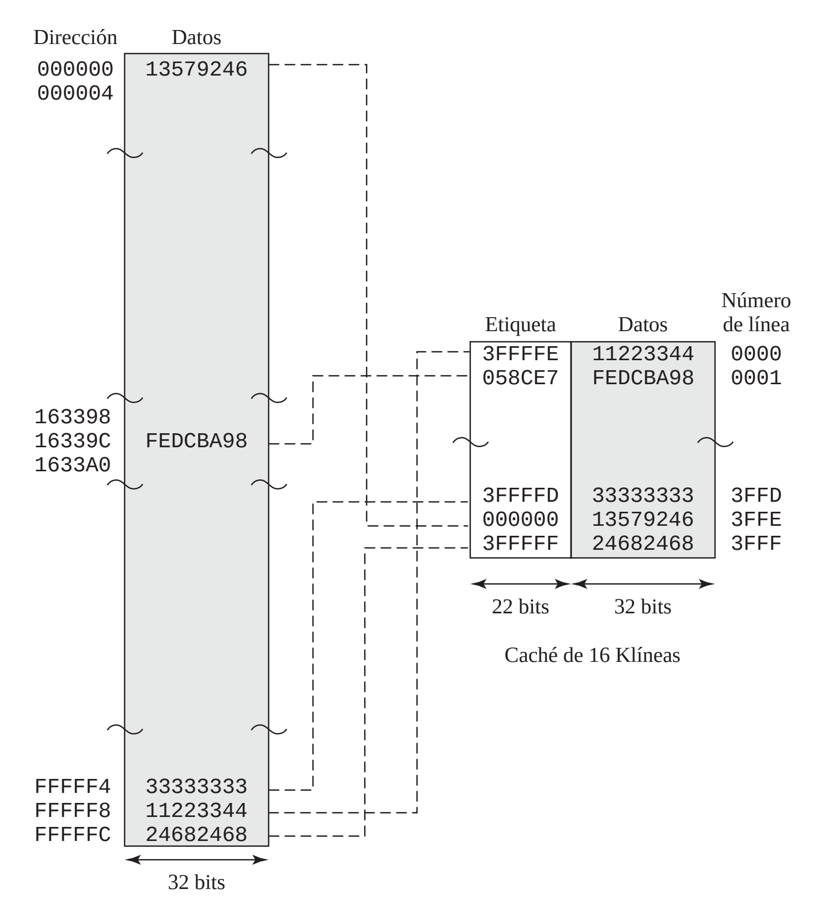


Figura 4.9. Organización de caché totalmente asociativa [HWAN93].



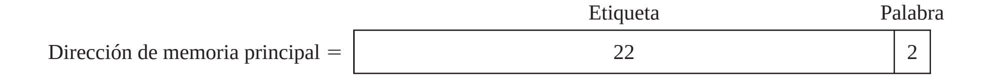


Figura 4.10. Ejemplo de correspondencia asociativa.

Con la correspondencia asociativa hay flexibilidad para que cualquier bloque sea reemplazado cuando se va a escribir uno nuevo en la caché. Los algoritmos de reemplazo o sustitución, discutidos más adelante en esta sección, se diseñan para maximizar la tasa de aciertos. La principal desventaja de la correspondencia asociativa es la compleja circuitería necesaria para examinar en paralelo las etiquetas de todas las líneas de caché.

Correspondencia asociativa por conjuntos. La correspondencia asociativa por conjuntos es una solución de compromiso que recoge lo positivo de las correspondencias directa y asociativa, sin presentar sus desventajas. En este caso, la caché se divide en v conjuntos, cada uno de k líneas. Las relaciones que se tienen son:

$$m = v \times k$$
$$i = j \text{ módulo } v$$

donde

$i =$ número de conjunto de caché
$j =$ número de bloque de memoria principal
$m =$ número de líneas de la caché

En este caso se denomina correspondencia asociativa por conjuntos de *k* vías. Con la asignación asociativa por conjuntos, el bloque B_j puede asignarse en cualquiera de las *k* líneas del conjunto *i*. En este caso, la lógica de control de la caché interpreta una dirección de memoria como tres campos: etiqueta, conjunto y palabra. Los *d* bits de conjunto especifican uno de entre $v = 2^d$ conjuntos. Los *s* bits de los campos de etiqueta y de conjunto especifican uno de los 2^s bloques de memoria principal. La Figura 4.11 muestra la lógica de control de la caché. Con la correspondencia totalmente asociativa, la etiqueta en una dirección de memoria es bastante larga y debe compararse con la etiqueta de cada línea en la caché. Con la correspondencia asociativa por conjuntos de *k* vías, la etiqueta de una dirección de memoria es mucho más corta y se compara solo con las *k* etiquetas dentro de un mismo conjunto. Resumiendo:

- Longitud de las direcciones $= (s + w)$ bits
- Número de unidades direccionables $= 2^{s+w}$ palabras o bytes
- Tamaño de bloque = tamaño de línea $= 2^w$ palabras o bytes
- Número de bloques en memoria principal $= \dfrac{2^{s+w}}{2^w} = 2^s$
- Número de líneas en el conjunto $= k$
- Número de conjuntos $= v = 2^d$
- Número de líneas en caché $= kv = k \times 2^d$
- Tamaño de la etiqueta $= (s - d)$ bits

Ejemplo 4.2c. La Figura 4.12 muestra nuestro ejemplo utilizando correspondencia asociativa por conjuntos con dos líneas por cada conjunto, denominada asociativa por conjuntos de dos vías[6]. El número de conjunto, de 13 bits, identifica un único conjunto de dos líneas dentro de la caché. También da el número, módulo 2^{13}, del bloque de memoria principal. Esto determina la asignación de bloques en líneas. Así, los bloques de memoria principal 000000, 008000..., FF8000, se hacen corresponder al conjunto 0 de la caché. Cualquiera de dichos bloques puede cargarse en alguna de las dos líneas del conjunto. Obsérvese que no hay dos bloques que se hagan corresponder al mismo conjunto de la caché que tengan el mismo número de etiqueta. Para una operación de lectura, el número de conjunto, de 13 bits, se utiliza para determinar qué conjunto de dos líneas va a examinarse. Ambas líneas del conjunto se examinan buscando una coincidencia con el número de etiqueta de la dirección a la que se va a acceder.

---

[6] En la Figura 4.12, la etiqueta de nueve bits se representa mediante un número hexadecimal de tres dígitos. El dígito más significativo tiene una longitud efectiva de solo un bit. El campo de conjunto+palabra, de quince bits, de la dirección de memoria principal, está representado en la figura con números de cuatro dígitos hexadecimales; de los cuales, el más significativo tiene una longitud efectiva de solo tres bits.

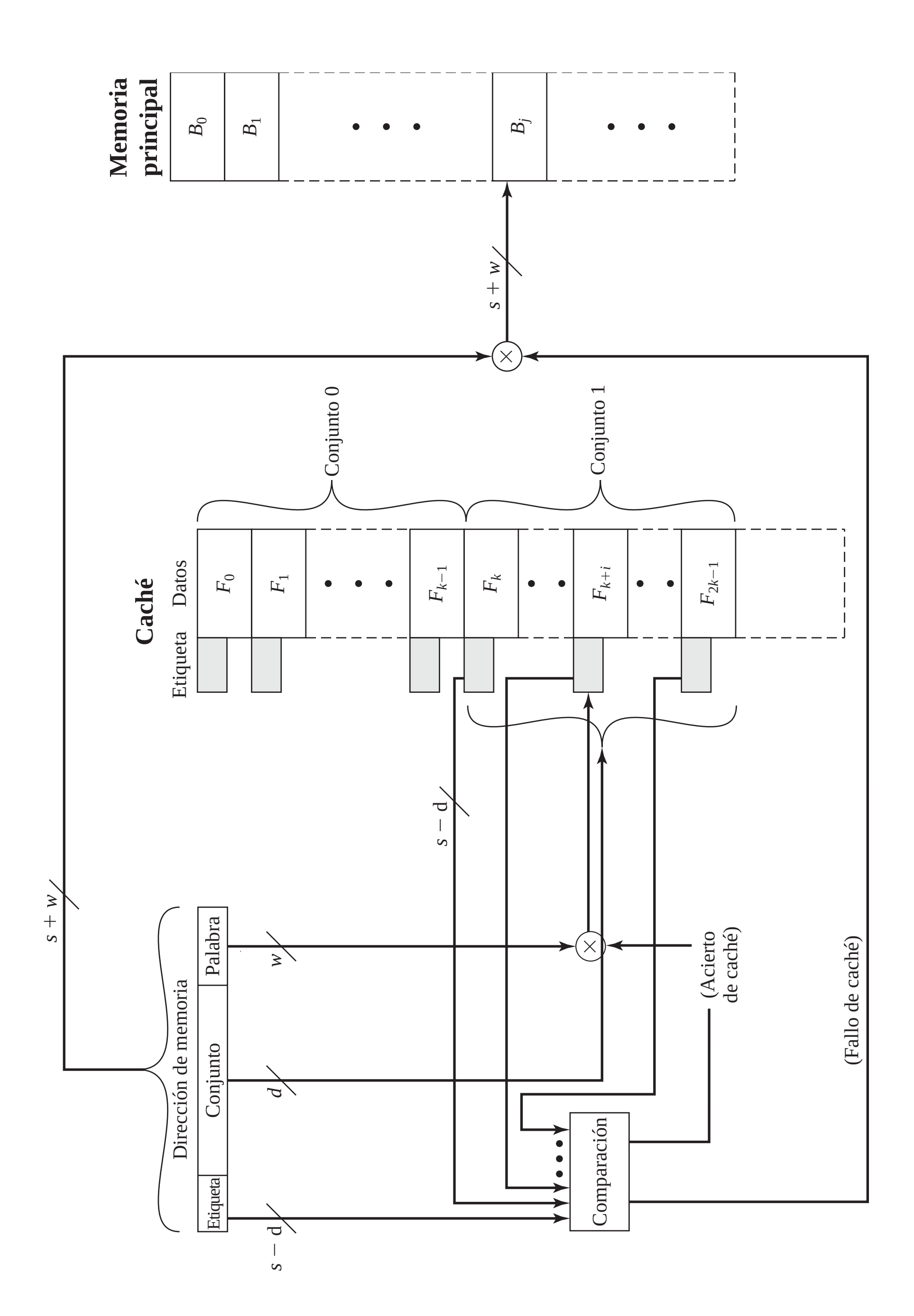


Figura 4.11. Estructura de caché asociativa por conjuntos de *k* vías.

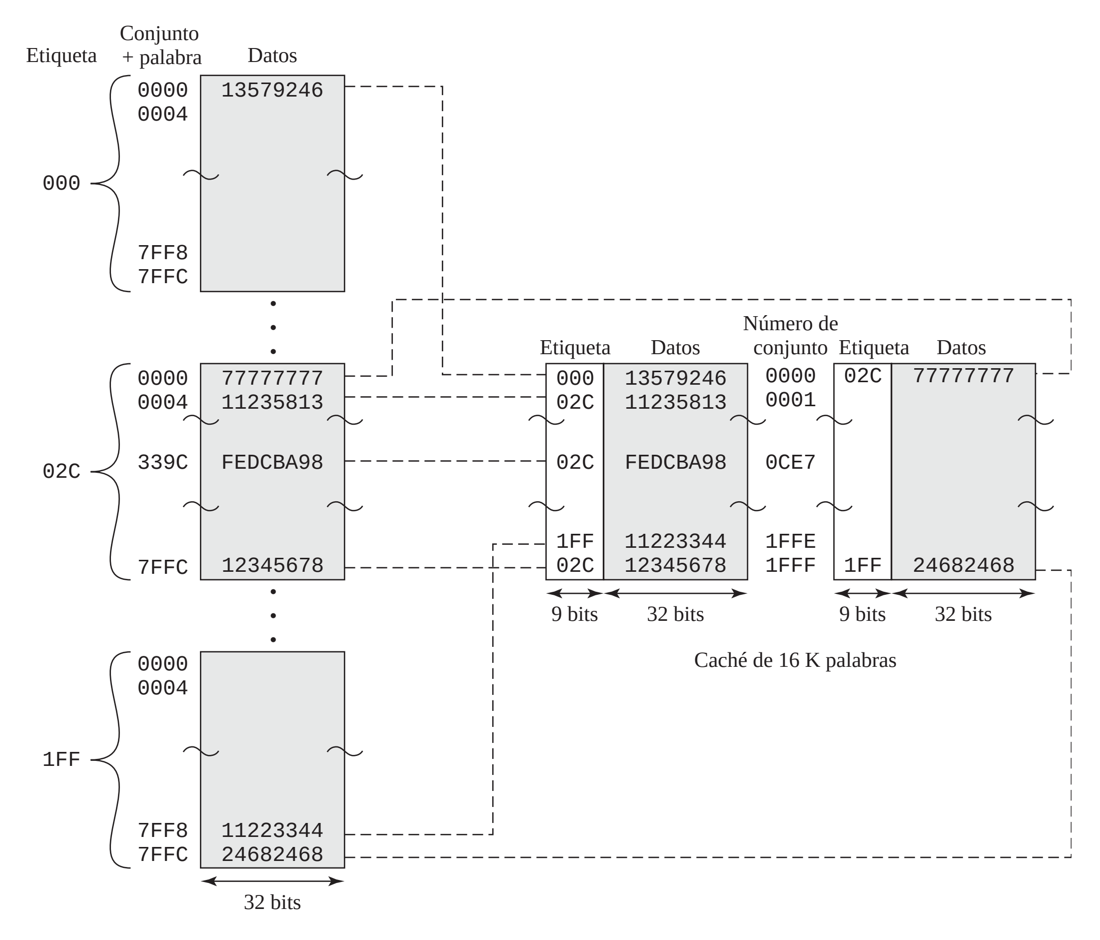


Figura 4.12. Ejemplo de correspondencia asociativa por conjuntos de dos vías.

En el caso extremo de $v = m$, $k = 1$, la técnica asociativa por conjuntos se reduce a la correspondencia directa, y para $v = 1$, $k = m$, se reduce a la totalmente asociativa. El uso de dos líneas por conjunto ($v = m/2$, $k = 2$) es el caso más común, mejorando significativamente la tasa de aciertos respecto de la correspondencia directa. La asociativa por conjuntos de *cuatro vías* ($v = m/4$, $k = 4$) produce una modesta mejora adicional con un coste añadido relativamente pequeño [MAYB84, HILL89]. Un incremento adicional en el número de líneas por conjunto tiene poco efecto.

ALGORITMOS DE SUSTITUCIÓN

Una vez que se ha llenado la caché, para introducir un nuevo bloque debe sustituirse uno de los bloques existentes. Para el caso de correspondencia directa, solo hay una posible línea para cada bloque particular y no hay elección posible. Para las técnicas asociativas se requieren algoritmos de sustitución. Para conseguir alta velocidad, tales algoritmos deben implementarse en hardware. Se han probado diversos algoritmos; mencionaremos cuatro de los más comunes. El más efectivo es probablemente el denominado "utilizado menos recientemente" (LRU, *least-recently used*): se sustituye el bloque que se ha mantenido en la caché por más tiempo sin haber sido referenciado. Esto es fácil de implementar para la asociativa por conjuntos de dos vías. Cada línea incluye un bit USO. Cuando una línea es referenciada se pone a 1 su bit USO y a 0 el de la otra línea del mismo conjunto. Cuando va a transferirse un bloque al conjunto, se utiliza la línea cuyo bit USO es 0. Ya que estamos suponiendo que son más probables de referenciar las posiciones de memoria utilizadas más recientemente, el LRU debiera dar la mejor tasa de aciertos. Otra posibilidad es el *primero en entrar-primero en salir* (FIFO, *First-In-First-Out*): se sustituye aquel bloque del conjunto que ha estado más tiempo en la caché. El algoritmo FIFO puede implementarse fácilmente mediante una técnica cíclica (*round-robin*) o buffer circular. Otra posibilidad más es la del *utilizado menos frecuentemente* (LFU, *Least-Frequently Used*): se sustituye aquel bloque del conjunto que ha experimentado menos referencias. LFU podría implementarse asociando un contador a cada línea. Una técnica no basada en el grado de utilización consiste simplemente en coger una línea al azar (aleatoria) entre las posibles candidatas. Estudios realizados mediante simulación han mostrado que la sustitución aleatoria proporciona unas prestaciones solo ligeramente inferiores a un algoritmo basado en la utilización [SMIT82].

POLÍTICA DE ESCRITURA

Hay dos casos a considerar cuando se ha de reemplazar un bloque de la caché. Si el bloque antiguo de la caché no debe ser modificado, puede sobrescribirse con el nuevo bloque sin necesidad de actualizar el antiguo. Si se ha realizado al menos una operación de escritura sobre una palabra de la línea correspondiente de la caché, entonces la memoria principal debe actualizarse, rescribiendo la línea de caché en el bloque de memoria antes de transferir el nuevo bloque. Son posibles varias políticas de escritura con distintos compromisos entre prestaciones y coste económico. Hay dos problemas contra los que luchar. En primer lugar, más de un dispositivo puede tener acceso a la memoria principal. Por ejemplo, un módulo de E/S puede escribir/leer directamente en/de memoria. Si una palabra ha sido modificada solo en la caché, la correspondiente palabra de memoria no es válida. Además, si el dispositivo de E/S ha alterado la memoria principal, entonces la palabra de caché no es válida. Un problema más complejo ocurre cuando varios procesadores se conectan al mismo bus y cada uno de ellos tiene su propia caché local. En tal caso, si se modifica una palabra en una de las cachés, podría presumiblemente invalidar una palabra de otras cachés.

La técnica más sencilla se denomina **escritura inmediata**. Utilizando esta técnica, todas las operaciones de escritura se hacen tanto en caché como en memoria principal, asegurando que el contenido de la memoria principal siempre es válido. Cualquier otro módulo procesador-caché puede monitorizar el tráfico a memoria principal para mantener la coherencia en su propia caché. La principal desventaja de esta técnica es que genera un tráfico sustancial con la memoria que puede originar un cuello de botella. Una técnica alternativa, conocida como **postescritura**, minimiza las escrituras

en memoria. Con la postescritura, las actualizaciones se hacen solo en la caché. Cuando tiene lugar una actualización, se activa un bit ACTUALIZAR asociado a la línea. Después, cuando el bloque es sustituido, es (post) escrito en memoria principal si y solo si el bit ACTUALIZAR está activo. El problema de este esquema es que se tienen porciones de memoria principal que no son válidas, y los accesos por parte de los módulos de E/S tendrán que hacerse solo a través de la caché. Esto complica la circuitería y genera un cuello de botella potencial. La experiencia ha demostrado que el porcentaje de referencias a memoria para escritura es del orden del 15 por ciento [SMIT82]. Sin embargo, en aplicaciones de HPC, este porcentaje puede aproximarse al 33 por ciento (multiplicación de vectores) e incluso superar el 50 por ciento (en transposición de matrices).

Ejemplo 4.3. Considere una caché con un tamaño de línea de 32 bytes y una memoria principal que requiere 30 ns para transferir una palabra de 4 bytes. Para cualquier línea de caché que sea escrita al menos una vez antes de ser intercambiada, ¿qué número medio de veces debe haber sido escrita la línea antes del intercambio para que la postescritura resulte más eficiente que la escritura inmediata?

Para el caso de postescritura, cada línea modificada debe escribirse en memoria sólo una vez, al intercambiarse, invirtiendo $8 \times 30 = 240$ ns. En el caso de escritura inmediata, cada actualización de la línea exige escribir una palabra en memoria principal, tardando 30 ns. Por tanto, si en promedio, en las líneas en las que se escribe se realizan más de ocho escrituras antes de ser intercambiadas, la postescritura resulta más eficiente.

En una estructura de bus en la que más de un dispositivo (normalmente un procesador) tiene una caché y la memoria principal es compartida, se tropieza con un nuevo problema. Si se modifican los datos de una caché, se invalida no solamente la palabra correspondiente de memoria principal, sino también la misma palabra en otras cachés (si coincide que otras cachés tengan la misma palabra). Incluso si se utiliza una política de escritura inmediata, las otras cachés pueden contener datos no válidos. Un sistema que evite este problema se dice que mantiene la coherencia de caché. Entre las posibles aproximaciones a la coherencia de caché se incluyen:

- **Vigilancia del bus con escritura inmediata:** cada controlador de caché monitoriza las líneas de direcciones para detectar operaciones de escritura en memoria por parte de otros maestros del bus. Si otro maestro escribe en una posición de memoria compartida que también reside en la memoria caché, el controlador de caché invalida el elemento de la caché. Esta estrategia depende del uso de una política de escritura inmediata por parte de todos los controladores de caché.

- **Transparencia hardware:** se utiliza hardware adicional para asegurar que todas las actualizaciones de memoria principal, vía caché, quedan reflejadas en todas las cachés. Así, si un procesador modifica una palabra de su caché, esta actualización se escribe en memoria principal. Además, de manera similar se actualizan todas las palabras coincidentes de otras cachés.

- **Memoria excluida de caché:** solo una porción de memoria principal se comparte por más de un procesador, y esta se diseña como no transferible a caché. En un sistema de este tipo, todos los accesos a la memoria compartida son fallos de caché, porque la memoria compartida nunca se copia en la caché. La memoria excluida de caché puede ser identificada utilizando lógica de selección de chip o los bits más significativos de la dirección.

La coherencia de caché es un campo activo de investigación y será tratado con más detalle en el Capítulo 18.

TAMAÑO DE LÍNEA

Otro elemento de diseño es el tamaño de línea. Cuando se recupera y ubica en caché un bloque de datos, se recuperan no sólo la palabra deseada sino además algunas palabras adyacentes. A medida que aumenta el tamaño de bloque, la tasa de aciertos primero aumenta debido al principio de localidad, el cual establece que es probable que los datos en la vecindad de una palabra referenciada sean referenciados en un futuro próximo. Al aumentar el tamaño de bloque, más datos útiles son llevados a la caché. Sin embargo, la tasa de aciertos comenzará a decrecer cuando el tamaño de bloque se haga aún mayor y la probabilidad de utilizar la nueva información captada se haga menor que la de reutilizar la información que tiene que reemplazarse. Dos efectos concretos entran en juego:

- Bloques más grandes reducen el número de bloques que caben en la caché. Dado que cada bloque captado se escribe sobre contenidos anteriores de la caché, un número reducido de bloques da lugar a que se sobrescriban datos poco después de haber sido captados.
- A medida que un bloque se hace más grande, cada palabra adicional está más lejos de la requerida y por tanto es más improbable que sea necesaria a corto plazo.

La relación entre tamaño de bloque y tasa de aciertos es compleja, dependiendo de las características de localidad de cada programa particular, no habiéndose encontrado un valor óptimo definitivo. Un tamaño entre 8 y 64 bytes parece estar razonablemente próximo al óptimo [SMIT87, PRZY88, PRZY90, HAND98]. Para sistemas de HPC es más frecuente usar tamaños de línea de caché de 64 y 128 bytes.

NÚMERO DE CACHÉS

Cuando se introdujeron originalmente las cachés, un sistema tenía normalmente solo una caché. Más recientemente, se ha convertido en una norma el uso de múltiples cachés. Hay dos aspectos de diseño relacionados con este tema que son el número de niveles de caché, y el uso de caché unificada frente al de cachés separadas.

Cachés multinivel. Con el aumento de densidad de integración, ha sido posible tener una caché en el mismo chip del procesador: caché *on-chip*. Comparada con la accesible a través de un bus externo, la caché *on-chip* reduce la actividad del bus externo del procesador y por tanto reduce los tiempos de ejecución e incrementa las prestaciones globales del sistema. Cuando la instrucción o el dato requeridos se encuentran en la caché *on-chip,* se elimina el acceso al bus. Debido a que los caminos de datos internos al procesador son muy cortos en comparación con la longitud de los buses, los accesos a la caché *on-chip* se efectúan apreciablemente más rápidos que los ciclos de bus, incluso en ausencia de estados de espera. Además, durante este periodo el bus está libre para realizar otras transferencias.

La inclusión de una caché *on-chip* deja abierta la cuestión de si es además deseable una caché externa u *off-chip*. Normalmente la respuesta es afirmativa, y los diseños más actuales incluyen tanto caché *on-chip* como externa. La estructura más sencilla de este tipo se denomina caché de dos niveles, siendo la caché interna el nivel 1 (L1), y la externa el nivel 2 (L2). La razón por la que se inclu-

ye una caché L2 es la siguiente. Si no hay caché L2 y el procesador hace una petición de acceso a una posición de memoria que no está en la caché L1, entonces el procesador debe acceder a la DRAM o la ROM a través del bus. Debido a la lentitud usual del bus y a los tiempos de acceso de las memorias, se obtienen bajas prestaciones. Por otra parte, si se utiliza una caché L2 SRAM (RAM estática), entonces con frecuencia la información que falta puede recuperarse fácilmente. Si la SRAM es suficientemente rápida para adecuarse a la velocidad del bus, los datos pueden accederse con cero estados de espera, el tipo más rápido de transferencia de bus.

En la actualidad son destacables dos características de diseño de las cachés multinivel. En primer lugar, para el caso de una caché L2 externa, muchos diseños no usan el bus del sistema como camino para las transferencias entre la caché L2 y el procesador, sino que se emplea un camino de datos aparte para reducir el tráfico en el bus del sistema. En segundo lugar, gracias a la continua reducción de dimensiones de los componentes de los procesadores, es fácil encontrar procesadores que incorporan la caché L2 en el propio chip, con la consiguiente mejora de prestaciones.

La mejora potencial del uso de una caché L2 depende de las tasas de aciertos en ambas cachés L1 y L2. Varios estudios han demostrado que, en general, el uso de un segundo nivel de caché mejora las prestaciones (*véase* por ejemplo [AZIM92], [NOVI93], [HAND98]). No obstante, el uso de cachés multinivel complica todos los aspectos de diseño de la caché, incluyendo el tamaño, el algoritmo de sustitución y la política de escritura; en [HAND98] y [PEIR99] se discuten estos temas.

Con la creciente disponibilidad de superficie para caché en el propio chip, en la mayoría de los microprocesadores modernos se ha llevado la caché L2 al chip del procesador, y se añade una caché L3. Inicialmente, la caché L3 era accesible a través del bus externo, pero más recientemente los microprocesadores han incorporado una L3 *on-chip*. En cualquiera de los casos, añadir un tercer nivel de caché parece suponer una mejora de las prestaciones (*véase* por ejemplo [GHAI98]).

Caché unificada frente a cachés separadas. Cuando hicieron su aparición las cachés *on-chip*, muchos de los diseños contenían una sola caché para almacenar las referencias tanto a datos como a instrucciones. Más recientemente, se ha hecho normal separar la caché en dos: una dedicada a instrucciones y otra a datos.

Una caché unificada tiene varias ventajas potenciales:

- Para un tamaño dado de caché, una unificada tiene una tasa de aciertos mayor que una caché partida, ya que nivela automáticamente la carga entre captación de instrucciones y de datos. Es decir, si un patrón de ejecución implica muchas más captaciones de instrucciones que de datos, la caché tenderá a llenarse con instrucciones, y si el patrón de ejecución involucra relativamente más captaciones de datos, ocurrirá lo contrario.
- Solo se necesita diseñar e implementar una caché.

A pesar de estas ventajas, la tendencia es hacia cachés separadas, particularmente para máquinas super-escalares tales como el Pentium y el PowerPC, en las que se enfatiza la ejecución paralela de instrucciones y la precaptación de instrucciones futuras previstas. La ventaja clave del diseño de una caché partida es que elimina la competición por la caché entre el procesador de instrucciones y la unidad de ejecución. Esto es importante en diseños que cuentan con segmentación de cauce (*pipelining*) de instrucciones. Normalmente el procesador captará instrucciones anticipadamente y llenará un buffer con las instrucciones que van a ejecutarse. Supongamos ahora que se tiene una caché unificada de instrucciones/datos. Cuando la unidad de ejecución realiza un acceso a memoria para cargar y almacenar datos,

se envía la petición a la caché unificada. Si, al mismo tiempo, el precaptador de instrucciones emite una petición de lectura de una instrucción a la caché, dicha petición será bloqueada temporalmente para que la caché pueda servir primero a la unidad de ejecución permitiéndole completar la ejecución de la instrucción en curso. Esta disputa por la caché puede degradar las prestaciones, interfiriendo con el uso eficiente del cauce segmentado de instrucciones. La caché partida supera esta dificultad.

4.4. ORGANIZACIÓN DE LA CACHÉ EN EL PENTIUM 4 Y EL POWERPC

ORGANIZACIÓN DE CACHÉ EN EL PENTIUM 4

La evolución de la organización de la caché se observa claramente en la evolución de los microprocesadores de Intel (Tabla 4.4). El 80386 no incluía caché *on-chip*. El 80486 incluye una sola caché

Tabla 4.4. Evolución de la caché en los Intel.

Problema	Solución	Primer procesador en incluirla
Memoria externa más lenta que el bus del sistema.	Añadir caché externa usando tecnología de memoria más rápida.	386
El aumento de velocidad de los procesadores hace que el bus se convierta en un cuello de botella para el acceso a caché.	Trasladar la caché externa al mismo chip (caché *on-chip*), funcionando a la misma velocidad que el procesador.	486
La caché es pequeña debido a la disponibilidad de superficie *on-chip* limitada.	Añadir una caché L2 externa con tecnología más rápida que la empleada en la memoria principal.	486
Se produce contención cuando las unidades de precaptación de instrucciones y de ejecución necesitan acceder simultáneamente a la caché. En este caso, la unidad de precaptación se bloquea mientras la unidad de ejecución accede a los datos.	Crear cachés separadas de datos y de instrucciones.	Pentium
El incremento de velocidad del procesador hace que el bus externo sea un cuello de botella para el acceso a la caché L2.	Crear un bus externo específico o «puerta trasera» (BSB, *back-side bus*) que funcione a más velocidad que el bus principal. El BSB se dedica a la caché L2.	Pentium Pro
	Llevar la caché L2 al chip del procesador.	Pentium II
Algunas aplicaciones que trabajan con grandes bases de datos deben tener acceso rápido a cantidades ingentes de datos. Las cachés *on-chip* son demasiado pequeñas.	Añadir una caché L3 externa.	Pentium III
	Integrar la caché L3 *on-chip*.	Pentium 4

on-chip de 8 KB, utilizando un tamaño de línea de 16 bytes y una organización asociativa por conjuntos de cuatro vías. Todos los procesadores Pentium incluyen dos cachés L1 *on-chip,* una para datos y otra para instrucciones. Para el Pentium 4, la caché de datos es de 8 KB, utilizando un tamaño de línea de 64 bytes y una organización asociativa por conjuntos de cuatro vías. Posteriormente describiremos la caché de instrucciones del Pentium 4. El Pentium II incluye también una caché L2 que alimenta a las dos cachés L1. La caché L2 es asociativa por conjuntos de ocho vías, con una capacidad de 256 Kb y un tamaño de línea de 128 bytes. En el Pentium III se añadió una caché L3 que pasó a ser *on-chip* en las versiones avanzadas del Pentium 4.

La Figura 4.13 muestra un esquema simplificado de la estructura del Pentium 4, resaltando la ubicación de las tres cachés. El núcleo del procesador consta de cuatro componentes principales:

- **Unidad de captación/decodificación:** capta instrucciones en orden de la caché L2, las decodifica en una serie de micro-operaciones, y memoriza los resultados en la caché L1 de instrucciones.
- **Lógica de ejecución fuera-de-orden:** planifica la ejecución de micro-operaciones teniendo en cuenta las dependencias de datos y los recursos disponibles; de forma que puede planificarse la ejecución de micro-operaciones en un orden diferente del que fueron captadas de la secuencia de instrucciones. Si el tiempo lo permite, esta unidad planifica la ejecución especulativa de micro-operaciones que puedan necesitarse en el futuro.
- **Unidades de ejecución:** estas unidades ejecutan las micro-operaciones, captando los datos necesarios de la caché de datos L1, y almacenando los resultados temporalmente en registros.
- **Subsistema de memoria:** esta unidad incluye las cachés L2 y L3, y el bus del sistema, que se usa para acceder a la memoria principal cuando en las cachés L1 y L2 tiene lugar un fallo de caché, así como para acceder a los recursos de E/S del sistema.

A diferencia de la organización de los modelos Pentium anteriores, así como de la mayoría de los demás procesadores, la caché de instrucciones del Pentium 4 está situada entre la lógica de decodificación de instrucciones y el núcleo de ejecución. El motivo es que el Pentium, como discutiremos con más detalle en el Capítulo 14, decodifica o traduce sus instrucciones máquina a instrucciones más sencillas de tipo RISC, denominadas micro-operaciones. El uso de micro-operaciones sencillas, de longitud fija, posibilita la utilización de segmentación superescalar y de técnicas de planificación que mejoran las prestaciones. Sin embargo, las instrucciones máquina del Pentium son difíciles de decodificar, ya que tienen un número variable de bytes y muchas opciones diferentes. Se consigue mejorar las prestaciones si dicha decodificación se realiza independientemente de la lógica de planificación y de segmentación. Volveremos sobre este tema en el Capítulo 14.

La caché de datos emplea una política de postescritura: los datos se escriben en memoria principal solo cuando, habiendo sido actualizados, se eliminan de la caché. El procesador Pentium 4 puede configurarse dinámicamente para utilizar la política de escritura inmediata.

La caché L1 de datos se controla por dos bits de uno de los registros de control (*véase* la Tabla 4.5), rotulados CD (*Caché Disable:* inhabilitar caché) y NW (*Not Write Through:* no escritura inmediata). Hay también dos instrucciones del Pentium 4 que pueden utilizarse para controlar la caché: INVD invalida la memoria caché interna e indica que se invalide la caché externa (si la hay); WBINVD postescribe e invalida la caché interna, y entonces postescribe e invalida la caché externa.

Las dos cachés, L2 y L3, son asociativas por conjuntos de ocho vías, con un tamaño de línea de 128 bytes.

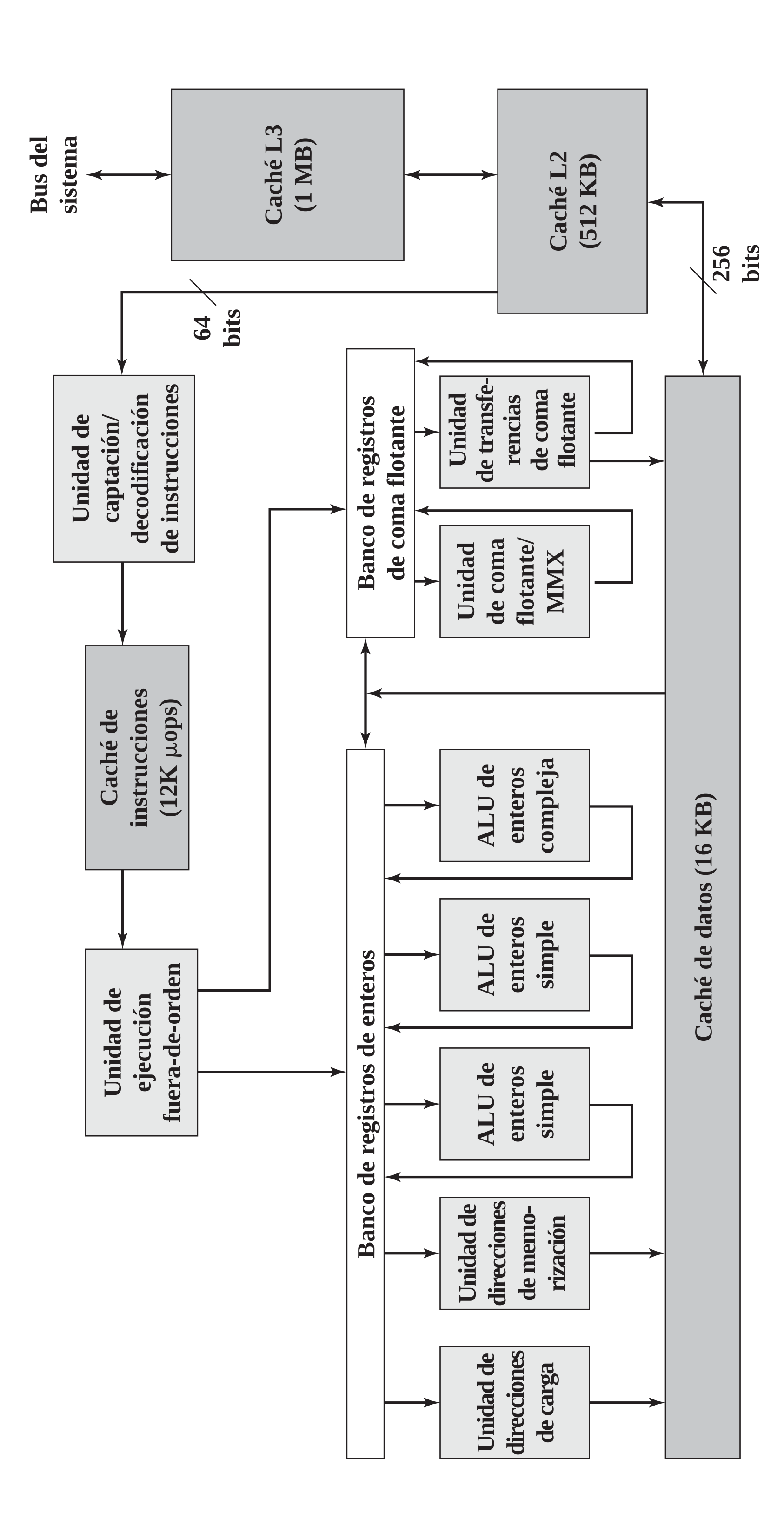


Figura 4.13. Diagrama de bloques del Pentium 4

Tabla 4.5. Modos de funcionamiento de la caché en el Pentium 4.

Bits de control		Modo de operación		
CD	**NW**	**Llenado de caché**	**Escrituras inmediatas**	**Invalidaciones**
0	0	Habilitado	Habilitadas	Habilitadas
1	0	Inhabilitado	Habilitadas	Habilitadas
1	1	Inhabilitado	Inhabilitadas	Inhabilitadas

NOTA: CD = 0; Nw = 1 es una combinación no válida.

ORGANIZACIÓN DE CACHÉ EN EL POWERPC

La organización de caché del PowerPC ha ido evolucionando paralelamente a la arquitectura global de la familia PowerPC, reflejando la búsqueda continua de mejores prestaciones que es el motor de todos los diseñadores de microprocesadores.

La Tabla 4.6 muestra esta evolución. El modelo original, el 601, incluye una sola caché de código/datos de 32 KB, que es asociativa por conjuntos de ocho vías. El 603 emplea un diseño RISC más sofisticado pero tiene una caché más pequeña: 16 KB divididos en cachés separadas de datos y de instrucciones, ambas con una organización asociativa por conjuntos de dos vías. Como resultado, el 603 tiene aproximadamente las mismas prestaciones que el 601 pero un coste menor. En cada modelo posterior, el 604 y el 620, se va duplicando el tamaño de las cachés respecto de su predecesor. Los modelos G3 y G4 tienen el mismo tamaño de cachés L1 que el 620. El G5 proporciona 32 KB de caché de instrucciones y 64 KB para caché de datos.

La Figura 4.14 es un esquema simplificado de la organización del PowerPC G5, resaltando la ubicación de las dos cachés. Las unidades de ejecución fundamentales son dos unidades aritméticas y lógicas de enteros, que pueden ejecutar en paralelo, y dos unidades de coma flotante con sus propios registros y componentes de multiplicación, suma y división. La caché de instrucciones, que es de sololectura, alimenta a la unidad de instrucciones, cuyo funcionamiento discutiremos en el Capítulo 14.

Las cachés L1 son asociativas por conjuntos de ocho vías. La caché L2 es asociativa por conjuntos de dos vías, con capacidades de 256 K, 512 K, o 1 MB. El G5 admite una caché L3 externa de hasta 1MB, y está previsto incorporar una caché L3 *on-chip* en implementaciones avanzadas del G5.

Tabla 4.6. Cachés L1 internas en la familia PowerPC.

Modelo	Tamaño	Bytes/línea	Organización
PowerPC 601	1 32-KB	32	Asociativa por conjuntos de 8 vías
PowerPC 603	2 8-KB	32	Asociativa por conjuntos de 2 vías
PowerPC 604	2 16-KB	32	Asociativa por conjuntos de 4 vías
PowerPC 620	2 32-KB	64	Asociativa por conjuntos de 8 vías
PowerPC G3	2 32-KB	64	Asociativa por conjuntos de 8 vías
PowerPC G4	2 32-KB	32	Asociativa por conjuntos de 8 vías
PowerPC G5	1 32-KB 1 64-KB	32	Asociativa por conjuntos de 8 vías

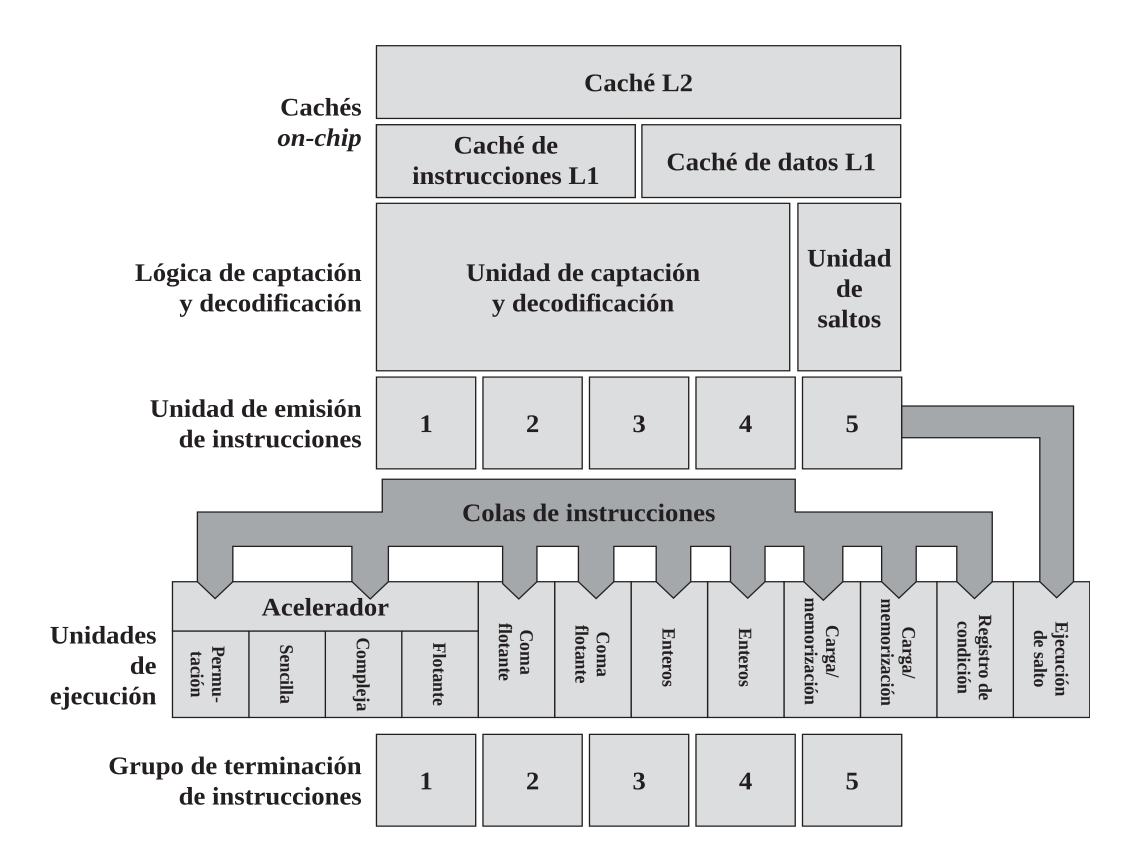


Figura 4.14. Diagrama de bloques del PowerPC G5.

4.5. LECTURAS RECOMENDADAS

Un tratamiento con profundidad del diseño de cachés se puede encontrar en [HAND98]. Una discusión sobre la organización de caché del Pentium 4 pueden encontrarse en [HINT01. Un artículo clásico que todavía merece la pena leer es [SMIT82], que revisa los distintos elementos del diseño de caché y presenta los resultados de múltiples análisis. Otro clásico interesante es [WILK65], que es probablemente el primer artículo que introduce el concepto de caché. [GOOD83] proporciona también un análisis útil del funcionamiento de la caché. Otro análisis que merece la pena es [BELL74]. En [AGAR89] se presenta un examen detallado de diversos aspectos de diseño de caché relacionados con multiprogramación y multiprocesamiento. [HIGB90] proporciona un conjunto de fórmulas sencillas que pueden utilizarse para estimar las prestaciones de una caché en función de varios parámetros.

AAGAR89 AGARWAL, A.: *Analysis of Cache Performance for Operating Systems and Multiprogramming.* Boston. Kluwer Academic Publishers, 1989.

BELL74 BELL, J.; CASASENT, D. y BELL, C.: «An Investigation into Alternative Cache Organizations». *IEEE Transactions on Computers,* abril 1974. **http://research.microsoft.com/users/GBell/gbvita.htm.**

GOOD83 GOODMAN, J.: «Using Cache Memory to Reduce Processor-Memory Bandwidth». *Proceedings, 10th Annual International Symposium on Computer Architecture,* 1983. Reprinted in [HILL00].

HAND98 HANDY, J.: *The Cache Memory Book.* San Diego. Academic Press, 1993.

HIGB90 HIGBIE, L.: «Quick and Easy Cache Performance Analysis». *Computer Architecture News,* junio 1990.

HINT01 HINTON, G., *et al.:* «The Microarchitecture of the Pentium 4 Processor». *Intel Technology Journal,* Q1 2001. **http://developer.intel.com/technology/itj/**

SMIT82 SMITH, A.: «Cache Memories». *ACM Computing Surveys,* septiembre, 1992.

WILK65 WILKES, M.: «Slave memories and dynamic storage allocation». *IEEE Transactions on Electronic Computers,* abril, 1965. Reimpreso en [HILL00].

4.6. PALABRAS CLAVE, PREGUNTAS DE REPASO Y PROBLEMAS

PALABRAS CLAVE

acceso aleatorio	caché separada o partida	fallo de caché
acceso directo	caché unificada	jerarquía de memoria
acceso secuencial	computación de altas prestaciones (HPC)	línea de caché
acierto de caché	conjunto de caché	localidad
algoritmo de sustitución	correspondencia asociativa	localidad espacial
caché de datos	correspondencia asociativa por conjuntos	localidad temporal
caché de instrucciones	correspondencia directa	memoria caché
caché l3	escritura inmediata	postescritura
caché L1	escritura única	tasa de aciertos
caché L2	etiqueta	tiempo de acceso
caché multinivel		

PREGUNTAS DE REPASO

4.1. ¿Qué diferencias hay entre acceso secuencial, acceso directo y acceso aleatorio?

4.2. ¿Cuál es la relación general entre tiempo de acceso, coste y capacidad de memoria?

4.3. ¿Cómo se relaciona el principio de localidad con el uso de múltiples niveles de memoria?

4.4. ¿Qué diferencias existen entre las correspondencias directa, asociativa y asociativa por conjuntos?

4.5. Para una caché con correspondencia directa, una dirección de memoria principal es vista como tres campos. Enumere y defina estos campos.

4.6. Para una caché con correspondencia asociativa, una dirección de memoria principal es vista como dos campos. Enumere y defina estos campos.

4.7. Para una caché con correspondencia asociativa por conjuntos, una dirección de memoria principal es vista como tres campos. Enumere y defina estos campos.

4.8. ¿Qué diferencia hay entre localidad espacial y localidad temporal?

4.9. En general, ¿cuáles son las estrategias para explotar la localidad espacial y la localidad temporal?

PROBLEMAS

4.1. Una caché asociativa por conjuntos consta de 64 líneas divididas en conjuntos de 4 líneas. La memoria principal contiene 4K bloques de 128 palabras cada uno. Muestre el formato de direcciones de memoria principal.

4.2. Una caché asociativa por conjuntos de dos vías tiene líneas de 16 bytes y una capacidad total de 8 KB. La memoria principal, de 64 MB, es direccionable por bytes. Muestre el formato de las direcciones de memoria principal.

4.3. Para las direcciones hexadecimales de memoria principal: 111111, 666666, BBBBBB; muestre en formato hexadecimal la siguiente información:

a) Los valores de etiqueta, línea y palabra para una caché con correspondencia directa, utilizando el formato de la Figura 4.8.

b) Los valores de etiqueta y de palabra para una caché asociativa, utilizando el formato de la Figura 4.10.

c) Los valores de etiqueta, conjunto y palabra para una caché asociativa por conjuntos de dos vías, utilizando el formato de la Figura 4.12.

4.4. Indique los siguiente valores:

a) Para la caché directa del ejemplo la Figura 4.8: la longitud de la dirección, el número de unidades direccionables, el tamaño de bloque, el número de bloques en memoria principal, el número de líneas en caché y el tamaño de la etiqueta.

b) Para la caché asociativa del ejemplo la Figura 4.10: la longitud de la dirección, el número de unidades direccionables, el tamaño de bloque, el número de bloques en memoria principal, el número de líneas en caché y el tamaño de la etiqueta.

c) Para la caché asociativa por conjuntos del ejemplo la Figura 4.12: la longitud de la dirección, el número de unidades direccionables, el tamaño de bloque, el número de bloques en memoria principal, el número de líneas en un conjunto, el número de líneas en caché y el tamaño de la etiqueta.

4.5. Considere un microprocesador de 32 bits que tiene una caché *on-chip* de 16 KBytes asociativa por conjuntos de cuatro vías. Suponga que la caché tiene un tamaño de línea de cuatro palabras de 32 bits. Dibuje un diagrama de bloques de esta caché mostrando su organización y cómo se utilizan los diferentes campos de dirección para determinar un acierto/fallo de caché. ¿Dónde se asigna, dentro de la caché, la palabra de la posición de memoria ABCDE8F8?

4.6. Dadas las siguientes especificaciones para una memoria caché externa: asociativa por conjuntos de cuatro vías; tamaño de línea de dos palabras de 16 bits; capaz de albergar un total de 4K palabras de 32 bits de la memoria principal; utilizada con un procesador de 16 bits que emite direcciones de 24 bits. Diseñe la estructura de caché con toda la información pertinente y muestre cómo interpreta las direcciones del procesador.

4.7. El Intel 80486 tiene una caché unificada *on-chip.* Esta caché es de 8 KB y tiene una organización asociativa por conjuntos de cuatro vías y una longitud de bloque de cuatro palabras de 32 bits. La caché está estructurada en 128 conjuntos. Hay un único «bit de línea válida» y tres bits, B0, B1, y B2 (los bits de LRU), por línea. En un fallo de caché, el 80486 lee una línea de 16 bytes de memoria principal en una ráfaga de lectura de memoria a través del bus. Dibuje un diagrama simplificado de la caché, y muestre cómo son interpretados los diferentes campos de la dirección.

4.8. Considere una máquina con una memoria principal de 2^{16} bytes, direccionable por bytes, y un tamaño de bloque de 8 bytes. Suponga que con esta máquina se utiliza una caché de 32 líneas y correspondencia directa.

a) ¿Cómo se divide la dirección de memoria de 16 bits entre etiqueta, número de línea y número de byte?

b) ¿En qué líneas se almacenarían los bytes que se encuentran en las siguientes direcciones?
0001 0001 0001 1011
1100 0011 0011 0100
1101 0000 0001 1101
1010 1010 1010 1010

c) Suponga que se almacena en la caché el byte de dirección 0001 1010 0001 1010. ¿Cuáles son las direcciones de los bytes que se almacenan junto con él?

d) ¿Cuántos bytes de memoria pueden almacenarse en total en la caché?

e) ¿Por qué se almacenan también las etiquetas en la caché?

4.9. Para su caché *on-chip,* el Intel 486 utiliza un algoritmo de sustitución denominado pseudo LRU. Asociados con cada uno de los 128 conjuntos de cuatro líneas (etiquetadas L0, L1, L2, L3) hay tres bits, B0, B1, y B2. El algoritmo de sustitución opera así: cuando se debe sustituir una línea, la caché determinará primero si el uso más reciente fue de L0 y L1 o de L2 y L3. Entonces la caché determinará cuál de la pareja de bloques fue utilizado menos recientemente y lo marcará para sustituirlo. La figura 4.15 muestra la lógica asociada.

a) Especifique cómo se ponen los bits B0, B1 y B2, y cómo se utilizan estos en el algoritmo de sustitución de la Figura 4.15.

b) Muestre cómo el algoritmo del 80486 aproxima a un algoritmo LRU verdadero. *Sugerencia:* considere el caso en el que el orden de uso más reciente es L0, L2, L3, L1.

c) Demuestre que un algoritmo LRU verdadero requeriría seis bits por conjunto.

4.10. Una caché asociativa por conjuntos tiene un tamaño de bloque de cuatro palabras de 16 bits y un tamaño de conjunto de 2. La caché puede albergar un total de 4096 palabras. El tamaño de memoria principal que es transferible a caché es de 64K $\times$ 32 bits. Diseñe la estructura de caché y muestre cómo son interpretadas las direcciones del procesador.

4.11. Considere un sistema de memoria que emplea direcciones de 32 bits para direccionar a nivel de bytes, más una caché que usa un tamaño de línea de 64 bytes.

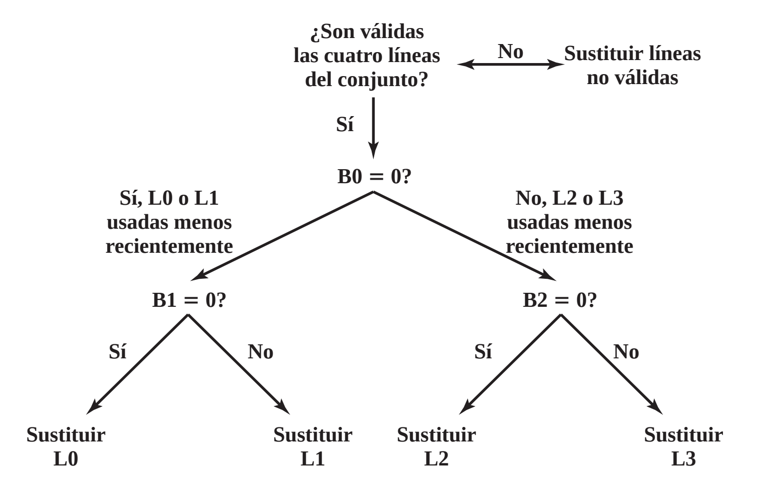


Figura 4.15. Estrategia de sustitución de la caché *on-chip* del 80486.

a) Suponga una caché con correspondencia directa, con un campo de etiqueta en la dirección de veinte bits. Muestre el formato de direcciones y determine los siguientes parámetros: número de unidades direccionables, número de bloques en memoria principal, número de líneas en caché y tamaño de la etiqueta.
b) Suponga una caché asociativa. Muestre el formato de las direcciones y determine los siguientes parámetros: número de unidades direccionables, número de bloques en memoria principal, número de líneas en caché y tamaño de la etiqueta.
c) Suponga una caché asociativa por conjuntos de 4 vías con un campo de etiqueta en la dirección de 9 bits. Muestre el formato de las direcciones y determine los siguientes parámetros: número de unidades direccionables, número de bloques en memoria principal, número de líneas en un conjunto, número de conjuntos en caché, número de lineas en caché y tamaño de la etiqueta.

4.12. Considere un computador con las siguientes características: un total de 1 MB de memoria principal; el tamaño de palabra es de un byte; el tamaño de bloque es de 16 bytes; y un tamaño de caché de 64 KB.

a) Para las direcciones de memoria principal: F0010, 01234, y CABBE, indique las correspondientes etiquetas, dirección de línea de caché y desplazamientos de palabras para una caché con correspondencia directa.
b) Indique dos direcciones cualesquiera de memoria principal con etiquetas diferentes que se correspondan con la misma línea para una caché con correspondencia directa.
c) Para las direcciones de memoria principal: F0010 y CABBE, indique los valores de etiqueta y de desplazamiento para una caché totalmente asociativa.
d) Para las direcciones de memoria principal: F0010 y CABBE, indique los valores de etiqueta, de conjunto de caché y de desplazamiento para una caché asociativa por conjuntos de dos vías.

4.13. Describa una técnica sencilla para implementar un algoritmo de sustitución LRU en una caché asociativa por conjuntos de cuatro vías.

4.14. Considere de nuevo el Ejemplo 4.3. ¿Cómo cambia el resultado si la memoria principal usa una capacidad de transferencia en bloques que tiene un tiempo de acceso de 30 ns a la primera de las palabras, y de 5 ns para cada una de las siguientes?

4.15. Considere el siguiente código:

for (i = 0; i < 20; i++)
for (j = 0; j < 10; j++)
a[i] = a[i] * j

a) Indique un ejemplo de localidad espacial en el código.
b) Indique un ejemplo de localidad temporal en el código.

4.16 Generalice las ecuaciones (4.1) y (4.2), del Apéndice 4A, a jerarquías de memoria de *N* niveles.

4.17. Un computador tiene una memoria principal de 32K palabras de 16 bits. Tiene también una caché de 4K palabras dividida en conjuntos de cuatro líneas con 64 palabras por línea. Suponga que la caché está inicialmente vacía. El procesador capta palabras de las posiciones 0, 1, 2..., 4351, en ese orden. Entonces repite esta secuencia de captación nueve veces más. La caché es diez veces más rápida que la memoria principal. Estime la mejora resultante por el uso de la caché. Suponga una política LRU para la sustitución de bloques.

4.18. Considere una caché de cuatro líneas con 16 bytes cada una. La memoria principal está dividida en bloques de 16 bytes. Es decir el bloque 0 tiene bytes con direcciones 0 a 15, y así sucesivamente. Considere ahora un programa que accede a memoria con la siguiente secuencia de direcciones:

Una vez: de 63 a 70.

Diez veces en un bucle: de 15 a la 32; y 80 a 95.

a) Suponga que la caché es de correspondencia directa. Los bloques de memoria 0, 4, etc., se asignan en la línea 1; los bloques 1, 5, etc., en la línea 2, y así sucesivamente. Calcule la tasa de aciertos.

b) Suponga ahora que la caché tiene una organización asociativa por conjuntos de 2 vías, on dos conjuntos de dos líneas cada uno. Los bloques con numeración par se asignan al conjunto 0 y los impares al conjunto 1. Calcule la tasa de aciertos para la caché asociatva por conjuntos de dos vías usando el esquema de sustitución LRU.

4.19. Considere un sistema de memoria con los siguientes parámetros:

$$T_c = 100 \text{ ns} \qquad C_c = 10^{-4} \text{ dólares/bit}$$
$$T_m = 1.200 \text{ ns} \qquad C_m = 10^{-5} \text{ dólares/bit}$$

a) ¿Cuál es el coste de una memoria principal de 1 MB?

b) ¿Cuál es el coste de una memoria principal de 1 MB utilizando la tecnología de la caché?

c) Si el tiempo de acceso efectivo es un 10 por ciento mayor que el tiempo de acceso de la caché, ¿cuál es la tasa de aciertos *H*?

4.20. a) Considere una caché L1 con un tiempo de acceso de 1 ns y una tasa de aciertos $H = 0{,}95$. Suponga que queremos cambiar el diseño de la caché (el tamaño, su organización) de manera que incrementemos *H* hasta 0,97, pero aumentando el tiempo de acceso a 1,5 ns. ¿Qué condiciones deben cumplirse para que este cambio suponga una mejora en las prestaciones?

b) Explique por qué el resultado tiene sentido intuitivamente.

4.21. Considere una caché de un solo nivel, con un tiempo de acceso de 2,5 ns, un tamaño de línea de 64 bytes y una tasa de aciertos $H = 0{,}95$. La memoria principal usa la capacidad de transferencia en bloques, con un tiempo de acceso de 50 ns para la primera palabra (4 bytes), y de 5 ns para cada una de las siguientes.

a) ¿Qué valor tiene el tiempo de acceso cuando hay un fallo de caché? Suponga que la caché espera hasta que la línea ha sido captada de memoria principal, para entonces ejecutar un acierto de caché.

b) Suponga que al incrementar el tamaño de línea a 128 bytes se incrementa *H* hasta 0,97. ¿Reduce esto el tiempo medio de acceso a memoria?

4.22. Un computador dispone de una caché, memoria principal, y un disco utilizado para memoria virtual. Cuando se referencia una palabra que está en la caché se requieren 20 ns para acceder a ella. Si está en memoria principal pero no en la caché se necesitan 60 ns para cargarla en la caché, y entonces se inicia de nuevo la referencia. Si la palabra no está en memoria principal se necesitan 12 ms para captarla de disco, seguidos de 60 ns para copiarla en la caché, comenzando entonces de nuevo la referencia. La tasa de aciertos de caché es de 0,9 y la de memoria principal de 0,6. ¿Cuál es, en nanosegundos, el tiempo medio necesario para acceder a una palabra referenciada en este sistema?

4.23. Considere una caché con un tamaño de línea de 64 bytes. Suponga que, en media, un 30 por ciento de las líneas de caché son modificadas. Una palabra consta de 8 bytes.

a) Suponga una tasa de fallos del 3 por ciento (tasa de aciertos de 0,97). Calcule la cantidad de tráfico de memoria principal en términos de bytes por instrucción, para políticas de escritura inmediata y de postescritura. Las lecturas de memoria principal a caché se realizan de línea en línea. Sin embargo, para la postescritura puede escribirse una sola palabra de caché a memoria principal.

b) Repita el apartado **a** para una tasa del 5 por ciento.

c) Repita el apartado **a** para una tasa del 7 por ciento.

d) ¿Qué conclusión puede extraerse de los resultados?

4.24. En el microprocesador Motorola 68020, un acceso a caché ocupa dos ciclos de reloj. El acceso a datos desde memoria principal, a través del bus, hasta el procesador, ocupa tres ciclos de reloj incluso cuando no se inserten estados de espera; los datos se entregan al procesador a la vez que se entregan a la caché.

a) Calcule la duración efectiva de un ciclo de memoria para una tasa de aciertos de 0,9 y una frecuencia de reloj de 16,67 MHz.

b) Repita los cálculos suponiendo que se insertan dos estados de espera de un ciclo por cada ciclo de memoria. ¿Qué conclusión puede extraerse de estos resultados?

4.25. Un procesador tiene un tiempo de ciclo de memoria de 300 ns y una velocidad de procesamiento de instrucciones de 1 MIPS. De media, cada instrucción necesita un ciclo de memoria del bus para captar la instrucción y otro para el operando involucrado.

a) Calcule la utilización del bus por parte del procesador.

b) Suponga que el procesador dispone de una caché de instrucciones con una tasa de aciertos asociada de 0,5. Determine el efecto que tiene sobre la utilización del bus.

4.26. Las prestaciones de un sistema de caché de solo un nivel, para una operación de lectura, puede caracterizarse mediante la ecuación:

$$T_a = T_c + (1 - H)\, T_m$$

Donde T_a es el tiempo de acceso medio, T_c es el tiempo de acceso a caché, T_m es el tiempo de acceso a memoria (de memoria a registro del procesador), y H es la tasa de aciertos. Para simplificar suponemos que la palabra en cuestión se carga en la caché en paralelo con su carga en el registro del procesador. Tiene la misma forma que la Ecuación (4.1).

a) Defina T_b = tiempo de transferencia de una línea entre caché y memoria principal, y W = fracción de referencias para escritura. Revise la ecuación anterior para que tenga en cuenta tanto las escrituras como las lecturas, usando una política de escritura inmediata.

b) Defina W_b como la probabilidad de una línea en caché haya sido modificada. Obtenga una ecuación para T_a, con una política de postescritura.

4.27. Para un sistema con dos niveles de caché, defina T_{c1} = tiempo de acceso a la caché del primer nivel; T_{c2} = tiempo de acceso a la caché del segundo nivel; T_m = tiempo de acceso a memoria; H_1 = tasa de aciertos de la caché del primer nivel; H_2 = tasa aciertos combinada del primer y el segundo nivel. Obtenga la ecuación de T_a para una operación de lectura.

4.28. Suponga el siguiente comportamiento frente a un fallo de caché: un ciclo de reloj para enviar una dirección a la memoria principal y cuatro ciclos de reloj para acceder a una palabra de 32 bits de la memoria principal y transferirla al procesador y a la caché.

a) Si el tamaño de línea de caché es de una palabra, ¿cuál es la penalización por fallo (es decir, el tiempo adicional necesario para una lectura cuando se produce un fallo de lectura)?

b) ¿Cuál es la penalización por fallo si el tamaño de línea de caché es de cuatro palabras, y se ejecuta una transferencia múltiple, no en ráfaga?

c) ¿Cuál es la penalización por fallo si el tamaño de línea de caché es de cuatro palabras y se ejecuta una transferencia, con un pulso de reloj para transferir cada palabra?

4.29. Para el diseño de caché del problema anterior, suponga que el incremento del tamaño de línea de una a cuatro palabras produce una disminución de la tasa de fallos de lectura del 3,2 por ciento al 1,1 por ciento. Para ambos casos, transferencia en ráfagas o no, ¿cuál es la penalización por fallo media, promediada sobre todas las lecturas, para los dos tamaños de línea indicados?

APÉNDICE 4A. PRESTACIONES DE LAS MEMORIAS DE DOS NIVELES

En este capítulo se ha hecho referencia a la caché que actúa como buffer entre la memoria principal y el procesador, creando una memoria interna de dos niveles. Esta arquitectura de dos niveles proporciona mejores prestaciones que una memoria comparable de un solo nivel, explotando una propiedad conocida como localidad, que analizamos más adelante en este apéndice.

El mecanismo de caché de la memoria principal es parte de la arquitectura del computador implementada en hardware y normalmente invisible para el sistema operativo. Además, hay otros dos ejemplos de memorias de dos niveles que también aprovechan la localidad y que se implementan, al menos parcialmente, en el sistema operativo: la memoria virtual y la caché de disco (Tabla 4.7). La

Tabla 4.7. Características de las memorias de dos niveles.

	Caché de memoria principal	Memoria virtual (paginación)	Caché de disco
Relaciones de tiempos de acceso típicas	5/1	$10^6 : 1$	$10^6 : 1$
Sistema de gestión de memoria	Implementado en hardware especial	Combinación de hardware y software del sistema	Software de sistema
Tamaño del bloque típico	4 a 128 bytes	64 a 4.096 bytes	64 a 4.096 bytes
Acceso del procesador al segundo nivel	Acceso directo	Acceso indirecto	Acceso indirecto

memoria virtual se verá en el Capítulo 8; la caché de disco queda fuera del alcance de este libro pero es examinada en [STAL05]. En este apéndice veremos algunas características de prestaciones de las memorias de dos niveles que son comunes a las tres aproximaciones mencionadas.

LOCALIDAD

La base para la mejora de prestaciones de una memoria de dos niveles es un principio conocido como **localidad de las referencias** [DENN68]. Este principio establece que las referencias a memoria tienden a formar agrupaciones (*clusters*). A lo largo de un período de tiempo largo las agrupaciones en uso cambian, pero durante periodos cortos el procesador trabaja fundamentalmente con agrupaciones fijas de referencias a memoria.

Intuitivamente, el principio de localidad tiene sentido. Considérense la siguiente secuencia de razonamientos:

1. Excepto por las instrucciones de bifurcación y de llamada, que constituyen solo una pequeña fracción de todas las instrucciones, la ejecución de un programa es secuencial. Por tanto, en la mayoría de los casos, la siguiente instrucción a captar sigue inmediatamente a la última captada.
2. Es raro tener una secuencia larga ininterrumpida de llamadas a procedimientos seguidas por la correspondiente secuencia de retornos. En su lugar, un programa queda confinado a una ventana bastante estrecha de profundidad o nivel de anidamiento de procedimientos. Así pues, a lo largo de un periodo de tiempo corto las referencias a instrucciones tienden a localizarse en unos cuantos procedimientos.
3. La mayoría de las construcciones iterativas constan de un número relativamente pequeño de instrucciones repetidas muchas veces. Durante una iteración, el procesamiento está por tanto confinado a una pequeña porción contigua del programa.
4. En muchos programas, gran parte del cálculo incumbe al procesamiento de estructuras de datos, tales como matrices o secuencias de registros. En muchos casos, las referencias sucesivas a estas estructuras de datos serán a unidades de datos ubicados próximos entre sí.

Esta secuencia de razonamientos ha sido confirmada en muchos estudios. En relación al punto 1, se han hecho diversos estudios para analizar el comportamiento de programas en lenguajes de alto nivel. La Tabla 4.8 recoge, de los estudios que se indican, resultados clave que miden la aparición de

Tabla 4.8. Frecuencia dinámica relativa de operaciones en lenguajes de alto nivel.

Estudio **Lenguaje** **Tipo de trabajo**	**[HUCK83]** **Pascal** **Científico**	**[KNUT71]** **FORTRAN** **Estudiante**	**[PATT82a]** **Pascal** **Sistema**	 **C** **Sistema**	**[TANE78]** **SAL** **Sistema**
Assign	74	67	45	38	42
Loop	4	3	5	3	4
Call	1	3	15	12	12
IF	20	11	29	43	36
GOTO	2	9	—	3	—
Otras	—	7	6	1	6

distintos tipos de sentencias durante la ejecución. El primero de los estudios sobre el comportamiento de lenguajes de programación, realizado por Knuth [KNU71], evaluó un conjunto de programas FORTRAN utilizados como ejercicios de estudiantes. Tanenbaum [TANE78] publicó medidas recopiladas de más de trescientos procedimientos utilizados en programas de sistemas operativos y escritos en un lenguaje que soporta programación estructurada (SAL). Patterson y Sequin [PATT82a] analizaron un conjunto de medidas tomadas de compiladores y programas para composición de textos, CAD, ordenación, y comparación de ficheros. Se estudiaron los lenguajes de programación C y Pascal. Huck [HUCK83] analizó cuatro programas ideados para una mezcla de cálculos científicos de uso general, incluyendo la transformada rápida de Fourier y la resolución de sistemas de ecuaciones diferenciales. Hay bastante coincidencia, en los resultados de esta mezcla de lenguajes y de aplicaciones, en que las instrucciones de bifurcación y de llamada representan solo una fracción de las sentencias ejecutadas durante el tiempo de vida de un programa. Estos estudios confirman pues la afirmación 1 anterior.

Con respecto a la afirmación 2, estudios aportados en [PATT85a] la confirman. Esto se ilustra en la Figura 4.16, que muestra el comportamiento de la pareja llamada-retorno. Cada llamada es representada mediante la línea hacia abajo y hacia la derecha, y cada retorno mediante la línea hacia arriba y a la derecha. En la figura se ha definido una *ventana* con profundidad igual a 5. Solo una secuencia de llamadas y retornos con variación de 6 en cualquier dirección hace que se traslade la

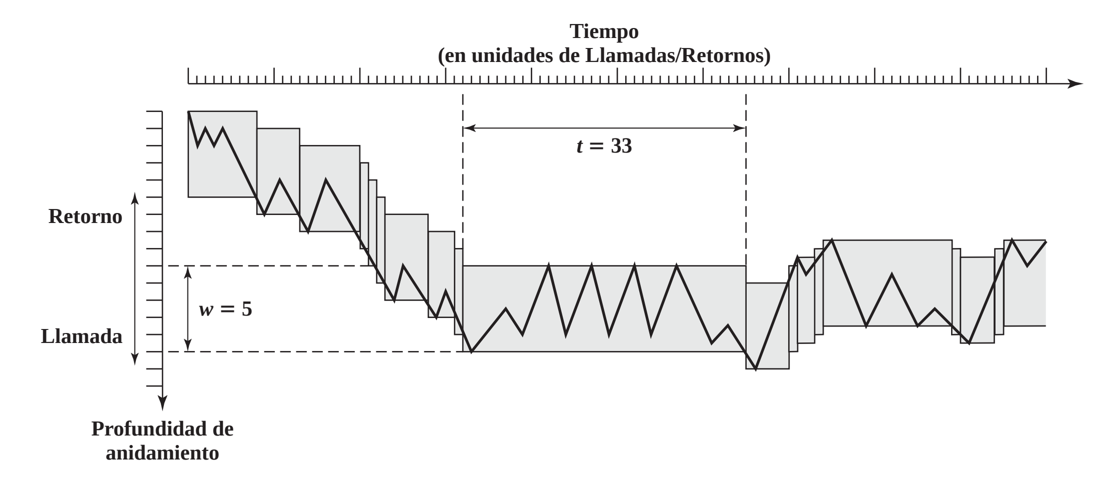


Figura 4.16 . Ejemplo de comportamiento de las llamadas/retornos en un programa.

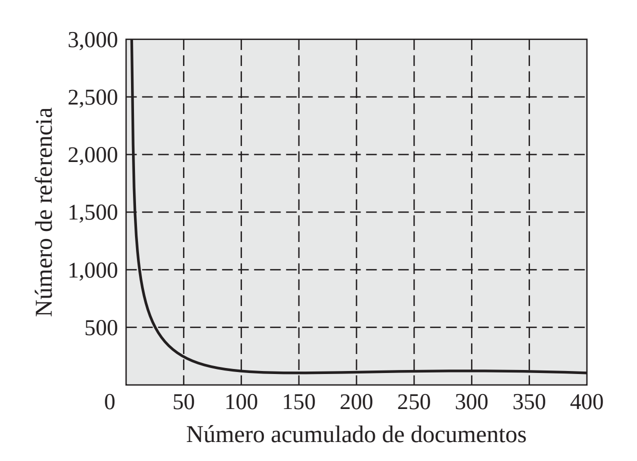


Figura 4.17. Localidad de referencias para páginas Web [BAEN97].

ventana. Como puede verse, el programa en ejecución puede mantenerse dentro de una ventana estacionaria por periodos de tiempo bastante largos. Un estudio por los mismos analistas de programas en C y en Pascal mostró que una ventana de profundidad 8 solo necesitaría desplazarse en menos del 1 por ciento de las llamadas o retornos [TAMI83].

El principio de localidad de las referencias continúa siendo validado en estudios más recientes. Por ejemplo, la Figura 4.17 muestra los resultados de un estudio sobre patrones de acceso a páginas web en un mismo sitio web.

En la literatura se distingue entre localidad espacial y temporal. La **localidad espacial** se refiere a la tendencia durante la ejecución a involucrar múltiples posiciones de memoria que estén agrupadas. La localidad espacial refleja la tendencia del procesador a acceder a las instrucciones secuencialmente y también la tendencia de los programas a acceder a posiciones de datos consecutivas, como por ejemplo cuando se procesa una tabla de datos. La **localidad temporal** hace referencia a la tendencia del procesador a acceder a posiciones de memoria que han sido utilizadas recientemente. Por ejemplo, cuando se ejecutan iteraciones de un bucle, el procesador ejecuta repetidamente el mismo conjunto de instrucciones.

Tradicionalmente, la localidad temporal se ha explotado manteniendo en memoria caché las instrucciones y los datos usados recientemente, y aprovechando la jerarquía de caché. Generalmente, la localidad espacial se explota usando bloques de caché más grandes e incorporando mecanismos de precaptación (captando objetos de uso anticipado) en la lógica de control de caché. Recientemente se ha investigado bastante para refinar estas técnicas al objeto de conseguir mayores prestaciones, pero las estrategias básicas siguen siendo las mismas.

FUNCIONAMIENTO DE LA MEMORIA DE DOS NIVELES

La propiedad de localidad puede ser aprovechada formando una memoria de dos niveles. La memoria del nivel superior (M1) es más pequeña más rápida, y más costosa (por bit) que la del nivel inferior (M2). M1 se utiliza como almacén temporal para una parte del contenido de la otra más grande, M2. Cuando se hace una referencia a memoria, se intenta acceder al elemento en M1. Si

tiene éxito, entonces tiene lugar un acceso rápido. Si no, se copia un bloque de posiciones de memoria de M2 a M1, y el acceso se hace vía M1. Debido a la localidad, una vez que el bloque es llevado a M1 habrá un número de accesos a posiciones de ese bloque, resultando un servicio rápido en su conjunto.

Para expresar el tiempo medio de acceso a un elemento, debemos considerar no solo las velocidades de los dos niveles de memoria, sino también la probabilidad de que una referencia dada se encuentre en M1. Tenemos:

$$\begin{aligned} T_s &= H \times T_1 + (1 - H) \times (T_1 + T_2) \\ &= T_1 + (1 - H) \times T_2 \end{aligned} \tag{4.1}$$

donde

T_s = tiempo de acceso medio (del sistema)

T_1 = tiempo de acceso de M1 (por ejemplo: caché, caché de disco)

T_2 = tiempo de acceso de M2 (por ejemplo: memoria principal, disco)

H = tasa de aciertos (fracción de veces que la referencia es encontrada en M1)

La Figura 4.2, al principio del capítulo, muestra el tiempo de acceso medio en función de la tasa de aciertos. Como puede verse, para un porcentaje de aciertos alto el tiempo de acceso total medio es mucho más próximo al de M1 que al de M2.

PRESTACIONES

Veamos algunos parámetros relevantes a la hora de evaluar un esquema de memoria de dos niveles. Consideremos primero el coste. Tenemos:

$$C_s = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2} \tag{4.2}$$

donde:

C_s = coste medio por bit de la combinación de dos niveles de memoria

C_1 = coste medio por bit de la memoria M1 del nivel superior

C_2 = coste medio por bit de la memoria M2 del nivel inferior

S_1 = tamaño de M1

S_2 = tamaño de M2

Nos gustaría que $C_s \approx C_2$. Dado que $C_1 >> C_2$, ello requiere que $S_1 << S_2$. La Figura 4.18 muestra dicha relación.

Consideremos ahora el tiempo de acceso. Para que una memoria de dos niveles suponga una mejora de prestaciones significativa, necesitamos tener T_s aproximadamente igual a T_1 ($T_s \approx T_1$). Dado que T_1 es mucho menor que T_2 ($T_1 << T_2$), se necesita una tasa de aciertos próxima a 1.

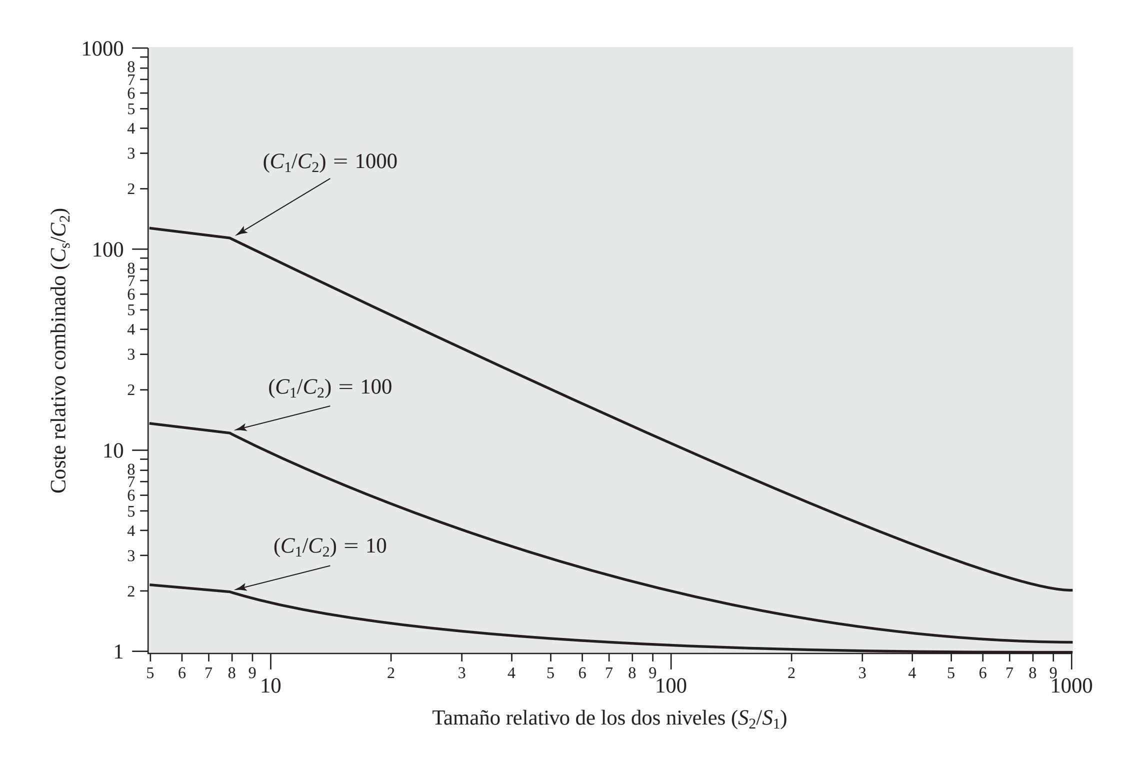


Figura 4.18. Relación entre el coste de memoria medio y el tamaño relativo de las memorias, para una memoria de dos niveles.

Así pues, nos gustaría que M1 fuera pequeña para mantener bajo el coste, y grande para mejorar la tasa de aciertos y en consecuencia las prestaciones. ¿Hay un tamaño de M1 que satisfaga razonablemente ambos requisitos? Esta pregunta la podemos responder con una serie de subpreguntas:

- ¿Qué valor de tasa de aciertos se necesita para que $T_s \approx T_1$?
- ¿Qué tamaño de M1 asegurará la tasa aciertos necesaria?
- ¿Satisface dicho tamaño el requisito del coste?

Para responder, consideremos la cantidad T_1/T_s, conocida como *eficiencia de acceso*. Es una medida de cuán próximo es el tiempo de acceso medio (T_s) al tiempo de acceso de M1 (T_1). De la ecuación (4.1) resulta:

$$\frac{T_1}{T_s} = \frac{1}{1 + (1 - H)\dfrac{T_2}{T_1}} \tag{4.3}$$

En la Figura 4.19 se representa T_1/T_s en función de la tasa de aciertos H, con la cantidad T_1/T_2 como parámetro. Normalmente el tiempo de acceso a la caché *on-chip* es entre 25 y 50 veces menor que el tiempo de acceso a memoria principal (es decir T_2/T_1 está entre 25 y 50), el tiempo de acceso

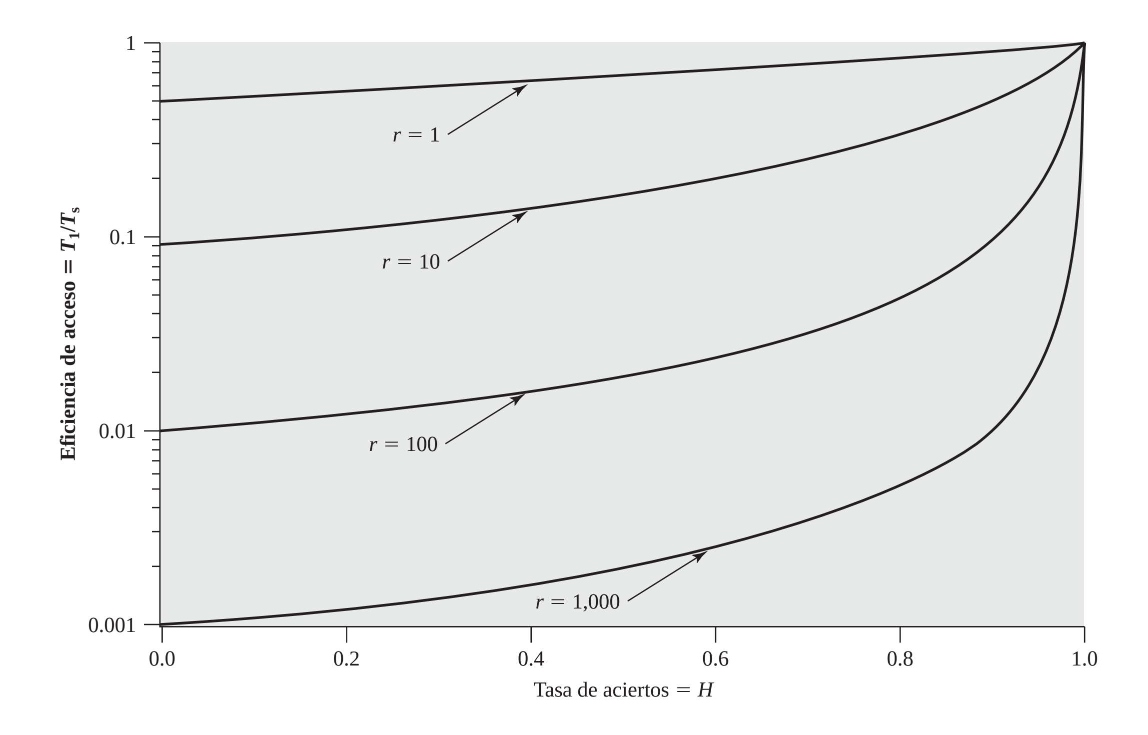


Figura 4.19. Eficiencia de acceso en función de la tasa de aciertos (r = T2/T1).

a caché externa (*off-chip*) es entre cinco y quince veces menor que el tiempo de acceso a memoria principal (es decir T_2/T_1 está entre 5 y 15)[7] y el acceso a memoria principal es del orden de 1 000 veces más rápido que el acceso a disco ($T_2/T_1 = 1\,000$). Por tanto, parece necesaria una tasa de aciertos entre 0,8 y 0,9 para satisfacer el requisito de prestaciones.

Ahora podemos plantear con mayor exactitud la cuestión referida al tamaño relativo de las memorias. ¿Es razonable una tasa de aciertos de 0,8 o superior con $S_1 << S_2$? Eso dependerá de diversos de factores, incluida la naturaleza del software que se ejecute y los detalles del diseño de la memoria de dos niveles. Lo más decisivo es, por supuesto, el grado de localidad. La Figura 4.20 sugiere el efecto que tiene la localidad sobre la tasa de aciertos. Claramente, si M_1 tiene el mismo tamaño que M_2, la tasa de aciertos será 1,0: todos los contenidos de M_2 están también siempre almacenados en M_1. Supongamos ahora que no hay localidad, esto es, que las referencias son completamente aleatorias. En este caso la tasa de aciertos sería una función estrictamente lineal del tamaño relativo de las memorias. Por ejemplo, si M_1 tiene la mitad de capacidad que M_2, en todo momento la mitad de los elementos de M_2 están también en M_1, y la tasa de aciertos será 0,5. En la práctica sin embargo existe cierto grado de localidad en las referencias a memoria. Los efectos de una localidad moderada y fuerte se indican en la figura.

Así pues, si la localidad es fuerte es posible conseguir una tasa de aciertos alta incluso con un tamaño relativamente pequeño de la memoria de nivel superior. Por ejemplo, numerosos estudios han

---

[7] Por ejemplo, para un Pentium 4 el tiempo de acceso a la caché *on-chip* es de 1 ns para la caché de datos, 2 ns para la caché de instrucciones y 3,5 ns para la caché L2; el tiempo de acceso a la memoria principal es de 30 ns. Para el Itanium 2, el tiempo de acceso a la caché *on-chip* es de 0,67 ns para la caché L1, 4 ns para la caché L2 y 8 ns para la caché L3.

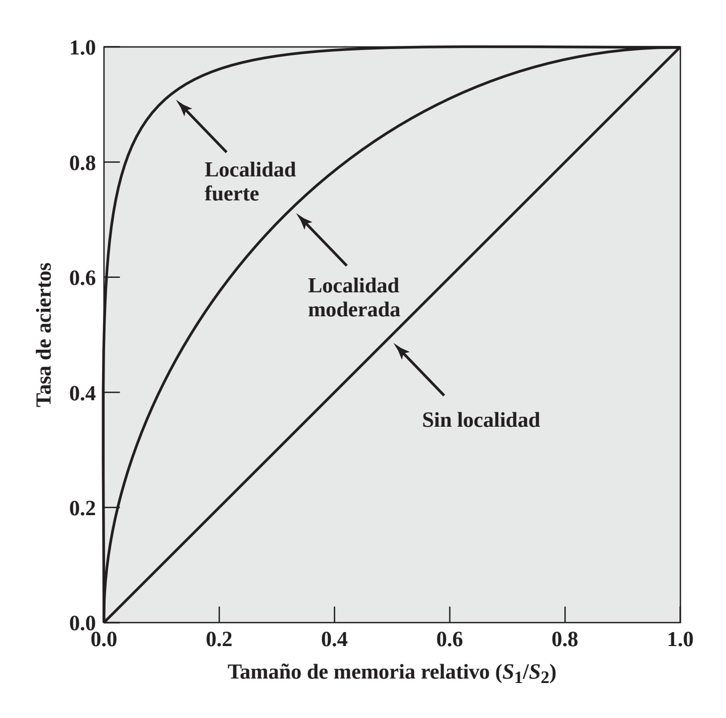


Figura 4.20. Tasa de aciertos en función del tamaño relativo de las memorias.

mostrado que tamaños de caché bastante pequeños producen tasas de aciertos por encima de 0,75, *con independencia del tamaño de la memoria principal* (*consúltese* por ejemplo [AGAR89], [PRZY88], [STRE83], y [SMIT82]). Generalmente es adecuada una caché de 1K a 128K palabras, mientras que una memoria principal actual se mueve en el rango de los gigabytes. Cuando consideremos la memoria virtual y la caché de disco citaremos otros estudios que confirman el mismo fenómeno, es decir que una M_1 relativamente pequeña produce un valor elevado de la tasa de aciertos gracias a la localidad.

Esto nos conduce a la última pregunta antes planteada: ¿satisface el tamaño relativo de las dos memorias el requisito del coste? La respuesta es claramente sí. Si para conseguir buenas prestaciones necesitamos poca capacidad para la memoria de nivel superior, el coste medio por bit de la memoria de dos niveles se aproximará a la del nivel inferior, que es más económica.

Observe que cuando hay implicada caché L2, o incluso cachés L2 y L3, el análisis es mucho más complejo. Consulte [PEIR99] y [HAND98] para una discusión sobre este tema.

CAPÍTULO 5

Memoria interna

PUNTOS CLAVE

- ➤ Las dos formas básicas de memorias semiconductoras de acceso aleatorio son la RAM dinámica (DRAM) y la RAM estática (SRAM). La SRAM es más rápida, más costosa y menos densa que la DRAM, y se usa para memorias cachés. La DRAM se usa para la memoria principal.
- ➤ En los sistemas de memoria es habitual utilizar técnicas de corrección de errores. Esto implica añadir bits redundantes, que se deducen a partir de los bits de datos, para formar un código de corrección de errores. Si ocurre un error en un bit el código lo detectará y, normalmente, lo corregirá.
- ➤ Para compensar la velocidad relativamente baja de la DRAM se han introducido diversas variantes con organizaciones DRAM avanzadas. Las dos más usuales son la DRAM síncrona y la DRAM RamBus. Ambas implican el uso del reloj del sistema para facilitar la transferencia de bloques de datos.

Este capítulo comienza con una revisión de los subsistemas de memoria principal semiconduto-ra, incluyendo las memorias ROM, DRAM y SRAM. Después nos centramos en las técnicas de control de errores utilizadas para incrementar la fiabilidad de las memorias. Posteriormente revisaremos las arquitecturas DRAM más avanzadas.

5.1. MEMORIA PRINCIPAL SEMICONDUCTORA

En computadores antiguos, la forma más común de almacenamiento de acceso aleatorio para la memoria principal consistía en una matriz de pequeños anillos ferromagnéticos denominados *núcleos*. Es por esto que la memoria principal recibía a menudo el nombre de *núcleo (core)*, un término que perdura en la actualidad. La llegada de la microelectrónica, y sus ventajas, acabó con las memorias de núcleos. Hoy en día es casi universal el uso de chips semiconductores para la memoria principal. En esta sección se exploran aspectos clave de esta tecnología.

ORGANIZACIÓN

El elemento básico de una memoria semiconductora es la celda de memoria. Aunque se utilizan diversas tecnologías electrónicas, todas las celdas de memoria semiconductora comparten ciertas propiedades:

- Presentan dos estados estables (o semiestables), que pueden emplearse para representar el 1 y el 0 binarios.
- Puede escribirse en ellas (al menos una vez) para fijar su estado.
- Pueden leerse para detectar su estado.

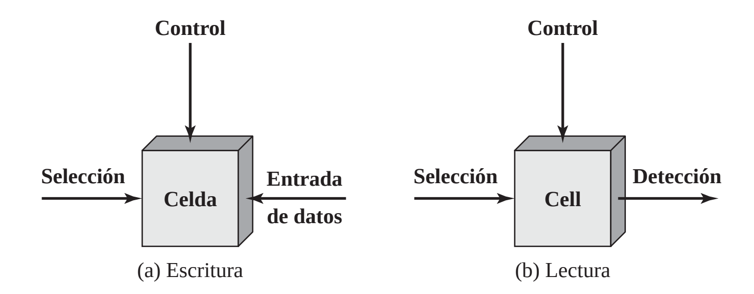


Figura 5.1. Funcionamiento de una celda de memoria.

La Figura 5.1 describe el funcionamiento de una celda de memoria. Lo más común es que la celda tenga tres terminales para transportar señales eléctricas. El terminal de selección, como su nombre indica, selecciona la celda para que pueda realizarse una operación de escritura o de lectura. El terminal de control indica si se trata de una lectura o de una escritura. Para la escritura, el tercer terminal proporciona la señal que fija el estado de la celda a uno o a cero. En una lectura, el tercer terminal se utiliza como salida del estado de la celda. Los detalles sobre estructura interna, funcionamiento y temporización de la celda de memoria, dependen de la tecnología específica de circuito integrado y, exceptuando un breve resumen, están más allá del alcance de este libro. Para nuestros propósitos, daremos por sentado que las celdas individuales pueden seleccionarse para operaciones de lectura y de escritura.

DRAM Y SRAM

Todos los tipos de memorias que estudiaremos en este capítulo son de acceso aleatorio. Es decir, las palabras individuales de la memoria son accedidas directamente mediante lógica de direccionamiento cableada interna.

La Tabla 5.1 lista los tipos principales de memorias semiconductoras. La más común es la denominada *memoria de acceso aleatorio* (RAM, *Random-Access Memory*). Este es, por supuesto, un mal uso del término ya que todas las memorias listadas en la tabla son de acceso aleatorio. Una característica distintiva de las RAM es que es posible tanto leer datos como escribir rápidamente nuevos datos en ellas. Tanto la lectura como la escritura se ejecutan mediante señales eléctricas.

La otra característica distintiva de una RAM es que es volátil. Una RAM debe estar siempre alimentada. Si se interrumpe la alimentación se pierden los datos. Así pues, las RAM pueden utilizarse solo como almacenamiento temporal. Las dos formas tradicionales de RAM utilizadas en los computadores son la DRAM y la SRAM.

RAM dinámica. Las tecnologías de RAM se dividen en dos variantes: dinámicas y estáticas. Una RAM dinámica (DRAM) está hecha con celdas que almacenan los datos como cargas eléctricas en condensadores. La presencia o ausencia de carga en un condensador se interpretan como el uno o el cero binarios. Ya que los condensadores tienen una tendencia natural a descargarse, las RAM dinámicas requieren refrescos periódicos para mantener memorizados los datos. El término *dinámica* hace referencia a esta tendencia a que la carga almacenada se pierda, incluso manteniéndola siempre alimentada.

Tabla 5.1. Tipos de memorias semiconductoras.

<table>
<tr><th>Tipo de memoria</th><th>Clase</th><th>Borrado</th><th>Mecanismos de escritura</th><th>Volatilidad</th></tr>
<tr><td>Memoria de acceso aleatorio (RAM)</td><td>Memoria de lectura/escritura</td><td>Eléctricamente por bytes</td><td>Eléctricamente</td><td>Volátil</td></tr>
<tr><td>Memoria de sólo lectura (ROM)</td><td rowspan="2">Memoria de sólo lectura</td><td rowspan="2">No posible</td><td>Mediante máscaras</td><td rowspan="5">No volátil</td></tr>
<tr><td>ROM programable (PROM)</td><td rowspan="4">Eléctricamente</td></tr>
<tr><td>PROM borrable (EPROM)</td><td rowspan="3">Memoria de sobre-todo-lectura</td><td>Luz ultravioleta, chip completo</td></tr>
<tr><td>Memoria FLASH</td><td>Eléctricamente, por bloques</td></tr>
<tr><td>PROM borrable eléctricamente (EEPROM)</td><td>Eléctricamente, por bytes</td></tr>
</table>

La Figura 5.2a muestra la estructura típica de una celda elemental de memoria DRAM, que memoriza un bit. La línea de direcciones se activa cuando se va a leer o a escribir el valor del bit de la celda. El transistor actúa como un conmutador que se cierra (permitiendo el paso de corriente) si se aplica tensión eléctrica a la línea de direcciones, y se abre (no fluye corriente) cuando la tensión aplicada es nula.

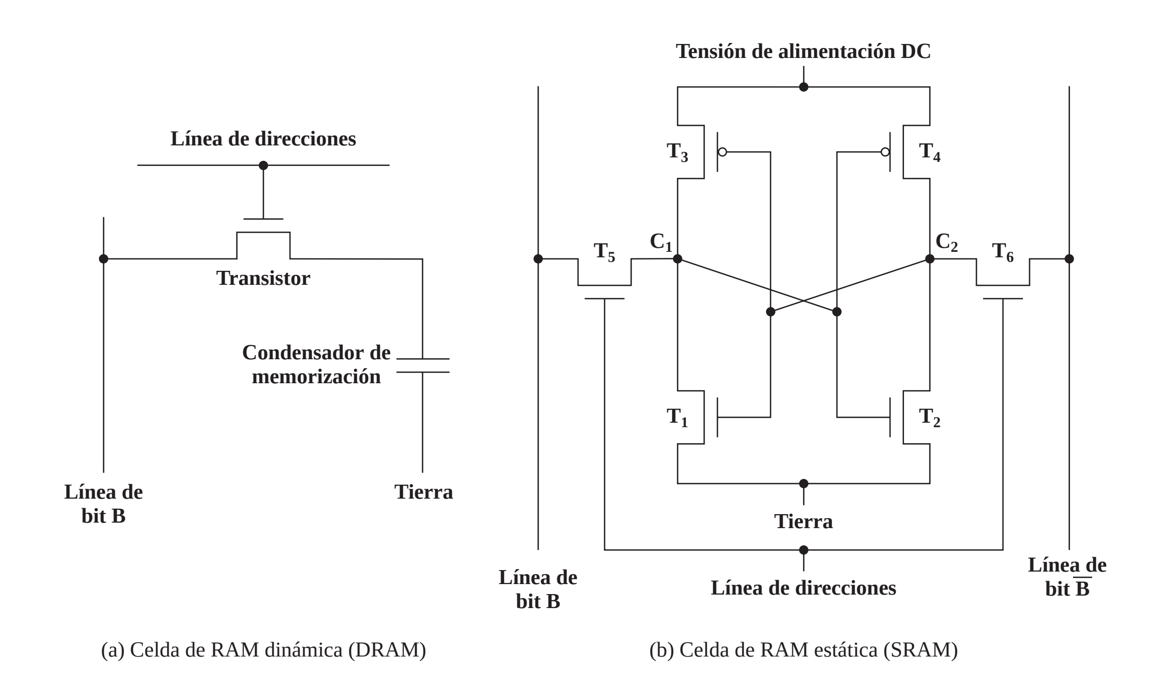


Figura 5.2. Estructuras típicas de celdas de memoria.

Para la operación de escritura se aplica un valor de tensión en la línea de bit; un valor de tensión alto representa un uno, y una tensión baja representa un cero. Se aplica entonces una señal a la línea de direcciones, permitiendo que se transfiera carga al condensador.

Para la operación de lectura, cuando se selecciona la línea de direcciones, el transistor entra en conducción y la carga almacenada en el condensador es transferida a la línea de bit y a un amplificador detector o amplificador de lectura. Este amplificador compara la tensión del condensador con un valor de referencia y determina si la celda contiene un uno lógico o un cero. La lectura de la celda descarga el condensador, cuya carga debe restablecerse para completar la operación.

Aunque la celda de DRAM se usa para almacenar un solo bit (0 ó 1), es un dispositivo esencialmente analógico. El condensador puede almacenar cualquier valor de carga dentro de un rango, y su comparación con un valor umbral determina si dicha carga se interpreta como uno o como cero.

RAM estática. En contraste con la dinámica, un RAM estática (SRAM) es un dispositivo digital, basado en los mismos elementos que se usan en el procesador. En una RAM estática, los valores binarios se almacenan utilizando configuraciones de puertas que forman biestables (*flip-flops*). Una descripción de los biestables puede verse en el Apéndice B. Una RAM estática retendrá sus datos en tanto se mantenga alimentada.

La Figura 5.2b muestra la estructura típica de una celda elemental de memoria SRAM. Cuatro transistores (T_1, T_2, T_3, T_4) están conectados en una configuración cruzada que produce estados lógicos estables. En el estado lógico 1, el punto C_1 está en alta y el C_2 en baja. En este estado T_1 y T_4 están en corte, y T_2 y T_3 están en conducción[1]. En el estado lógico 0, el punto C_1 está en baja y el C_2 en alta. En este estado T_1 y T_4 están en conducción, y T_2 y T_3 están en corte. Ambos estados son estables y se mantienen mientras se esté alimentando la celda con una tensión continua de «corriente directa» (dc). A diferencia de DRAM, no se necesita refrescar el dato para mantenerlo.

Al igual que en una celda DRAM, la línea de direcciones en la SRAM se emplea para abrir o cerrar un conmutador. La línea de direcciones controla en este caso dos transistores (T_5 y T_6). Cuando se aplica una señal a esta línea, los dos transistores entran en conducción, permitiendo la operación de lectura o de escritura. En una operación de escritura, el valor de bit deseado se aplica a la línea B, y su complemento se aplica a la línea $\overline{B}$. Esto fuerza a los cuatro transistores (T_1, T_2, T_3, T_4) al estado apropiado. En una operación de lectura, el valor de bit se lee de la línea B.

SRAM frente a DRAM. Tanto las RAM estáticas como las dinámicas son volátiles; es decir, debe aplicarse continuamente tensión de alimentación a la memoria para mantener los valores de los bits. Una celda de memoria RAM dinámica es más simple que una estática y en consecuencia más pequeña. Por tanto, las DRAM dinámicas son más densas (celdas más pequeñas = más celdas por unidad de superficie) y más económicas que las correspondientes SRAM. Por otra parte, una DRAM requiere de circuitería para realizar el refresco. En memorias grandes, el coste fijo de la circuitería de refresco se ve más que compensado por el menor coste de las celdas DRAM. Así pues, las DRAM tienden a ser las preferidas para memorias grandes. Un último detalle es que las SRAM son generalmente algo más rápidas que las dinámicas. Debido a estas características relativas, las SRAM se utilizan como memorias cachés (tanto *on-chip* como *off-chip*), y las DRAM para la memoria principal.

---

[1] Los círculos en los terminales de control de los transistores T3 y T4 representan el complemento o negación de las señales.

TIPOS DE ROM

Como sugiere su nombre, una **memoria de solo-lectura** (ROM, *Read-Only Memory*) contiene un patrón permanente de datos que no puede alterarse. Una ROM es no-volátil; es decir, no se requiere fuente de alimentación para mantener memorizados los valores de los bits. Aunque es posible leer de una ROM, no se pueden escribir nuevos datos en ella. Una aplicación importante de las ROM es la microprogramación, estudiada en la Parte Cuatro del libro. Otras aplicaciones potenciales son:

- Subrutinas de biblioteca para funciones de uso frecuente.
- Programas del sistema.
- Tablas de funciones.

Cuando se requiere un tamaño modesto, la ventaja de una ROM es que el programa o los datos estarían permanentemente en memoria principal y nunca sería necesario cargarlos desde un dispositivo de memoria secundaria.

Una ROM se construye como cualquier otro chip de circuito integrado, con los datos cableados en el chip durante el proceso de fabricación. Esto presenta dos problemas:

- La etapa de inserción de datos implica unos costes fijos relativamente grandes, tanto si se va a fabricar una o miles de copias de una misma ROM.
- No se permite un fallo. Si uno de los bits es erróneo, debe desecharse la tirada completa de chips de memoria ROM.

Cuando se necesitan solo unas pocas ROM con un contenido particular, una alternativa más económica es la **ROM programable** (PROM). Al igual que las ROM, las PROM son no volátiles y pueden grabarse solo una vez. Para la PROM, el proceso de escritura se lleva a cabo eléctricamente y puede realizarlo el suministrador o el cliente con posterioridad a la fabricación del chip original. Se requiere un equipo especial para el proceso de escritura o «programación». Las PROM proporcionan flexibilidad y comodidad. Las ROM siguen siendo atractivas para tiradas de producción de gran volumen.

Otra variante de memoria de solo lectura es la memoria de **sobre-todo lectura** (*read-mostly*), que es útil para aplicaciones en las que las operaciones de lectura son bastante más frecuentes que las de escritura, pero para las que se requiere un almacenamiento no volátil. Hay tres formas comunes de memorias de sobre-todo lectura: EPROM, EEPROM, y memorias *flash*.

La **memoria de solo-lectura programable y borrable** ópticamente (EPROM, *Erasable Programmable Read-Only Memory*) se lee y escribe eléctricamente como la PROM. Sin embargo, antes de la operación de escritura, todas las celdas de almacenamiento deben primero borrarse a la vez, mediante la exposición del chip encapsulado a radiación ultravioleta. Este proceso de borrado puede realizarse repetidas veces; cada borrado completo puede durar hasta veinte minutos. Así pues, las EPROM pueden modificarse múltiples veces y, al igual que las ROM y las PROM, retienen su contenido, en teoría indefinidamente. Para una capacidad similar, una EPROM es más costosa que una PROM, pero tiene como ventaja adicional la posibilidad de actualizar múltiples veces su contenido.

Una forma más interesante de memoria de sobre-todo lectura es la **memoria de solo-lectura programable y borrable eléctricamente** (EEPROM, *Electrically Erasable Programmable Read-Only*

Memory). Esta es una memoria de sobre-todo lectura en la que se puede escribir en cualquier momento sin borrar su contenido anterior; solo se actualiza el byte o bytes direccionados. La operación de escritura consume un tiempo considerablemente mayor que la de lectura; del orden de cientos de microsegundos por byte. La EEPROM combina la ventaja de ser no volátil, con la flexibilidad de ser actualizable *in situ*, utilizando las líneas de datos, de direcciones y de control de un bus ordinario. Las EEPROM son más costosas que las EPROM y también menos densas, admitiendo menos bits por chip.

Otra forma de memoria semiconductora es la **memoria flash** (denominada así por la velocidad con la que puede reprogramarse). Introducidas a mediados de los 1980, las memorias flash se encuentran, en coste y en funcionalidad, entre las EPROM y las EEPROM. Al igual que las EEPROM, las flash utilizan una tecnología de borrado eléctrico. Una memoria flash puede borrarse entera en uno o unos cuantos segundos, mucho más rápido que las EPROM. Además, es posible borrar solo bloques concretos de memoria en lugar de todo el chip. Las memorias flash deben su nombre a que su microchip está organizado de manera que cada una de sus secciones de celdas se borra mediante una única acción, de un golpe o *flash*. Sin embargo, las memorias flash no permiten borrar a nivel de byte. Al igual que las EPROM, las flash utilizan solo un transistor por bit, consiguiéndose las altas densidades (comparadas con las EEPROM) que alcanzan las EPROM.

LÓGICA DEL CHIP

Como otros circuitos integrados, las memorias semiconductoras vienen en chips encapsulados (Figura 2.7). Cada chip contiene una matriz de celdas de memoria.

En toda la jerarquía de memoria vimos que existen compromisos de velocidad, capacidad y coste. Estos compromisos existen también cuando consideramos la organización de las celdas de memoria y del resto de funciones lógicas de un chip de memoria. Para las memorias semiconductoras, uno de los aspectos fundamentales de diseño es el número de bits de datos que pueden ser leídos/escritos a la vez. En un extremo está la estructura en la que la disposición física de las celdas de la matriz es la misma que la disposición lógica (tal y como la percibe el procesador) de las palabras de memoria. La matriz está organizada en W palabras de B bits cada una. Por ejemplo, un chip de 16 Mb podría estar estructurado en 1 M palabras de 16 bits. En el otro extremo está la estructura denominada un-bit-por-chip, en la que los datos se escriben/leen por bits. A continuación describimos la estructura de un chip de memoria DRAM; la estructura de una ROM integrada es similar, aunque más sencilla.

La Figura 4.3 muestra una organización típica de DRAM de 16 Mb. En este caso se escriben o leen cuatro bits a la vez. Lógicamente, la matriz de memoria está estructurada en cuatro matrices cuadradas de 2048x2048 elementos. Son posibles varias disposiciones físicas. En cualquier caso, los elementos de la matriz conectan tanto a líneas horizontales (de fila) como a verticales (de columna). Cada línea horizontal conecta al terminal de Selección de cada celda en la correspondiente fila; y cada línea vertical conecta al terminal Entrada-Datos/Detección (*Data-In/Sense*) de cada celda en la correspondiente columna.

Las líneas de direcciones suministran la dirección de la palabra a seleccionar. Se requiere un total de $\log_2 W$ líneas. En nuestro ejemplo se necesitan once líneas de direcciones para seleccionar una de entre 2048 filas. Estas once líneas entran en un decodificador de filas, que tiene once líneas de entra-

da y 2048 de salida. La lógica del decodificador activa una única salida de entre las 2048, definida por el patrón de bits de las once líneas de entrada ($2^{11} = 2048$).

Otro grupo de once líneas de direcciones selecciona una de entre 2048 columnas, con cuatro bits por columna. Se utilizan cuatro líneas para la entrada y salida de cuatro bits, a y desde, un buffer de datos. Para la entrada (escritura), cada línea de bit se activa a uno o a cero de acuerdo con el valor de la correspondiente línea de datos. Para salida (lectura), el valor de cada línea de bit se pasa a través de un amplificador de lectura (término que emplearemos para referirnos al inglés *sense amplifier*) y se presenta en la correspondiente línea de datos. La línea de fila selecciona la fila de celdas que es utilizada para lectura o escritura.

Ya que en esta DRAM se escriben/leen solo cuatro bits, debe haber varias DRAM conectadas al controlador de memoria a fin de escribir/leer una palabra de datos en el bus.

Obsérvese que hay solo once líneas de direcciones (A0-A10), la mitad del número necesario para una matriz de 2048x2048. Esto se hace así para ahorrar en número de terminales. Las señales de las 22 líneas de direcciones necesarias se transforman con lógica de selección externa al chip y se multiplexan en once líneas de direcciones. Primero se proporcionan al chip once señales de dirección que definen la dirección de fila de la matriz, y después se presentan las otras once señales para la dirección de columna. Estas señales se acompañan por las de selección de dirección de fila *($\overline{RAS}$)* y de selección de dirección de columna *($\overline{CAS}$)* que temporizan el chip.

Los terminales de habilitación de escritura *($\overline{WE}$)* y de habilitación de la salida *($\overline{OE}$)* determinan si se realiza una operación de escritura o de lectura. Otros dos terminales, no mostrados en la Figura 5.3, son el de tierra (V_{ss}) y el tensión de alimentación (V_{cc}).

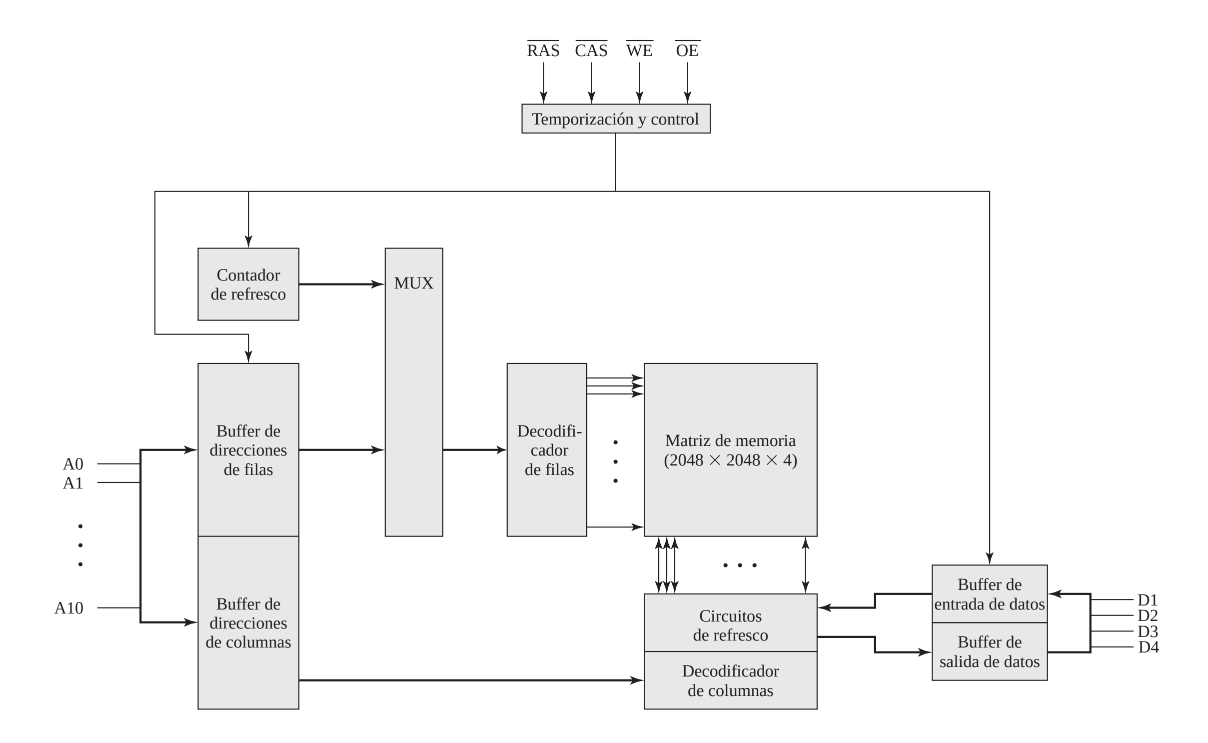


Figura 5.3. DRAM típica de 16 megabits (4M × 4).

Como comentario, el uso de direccionamiento multiplexado y de matrices cuadradas da lugar a que el tamaño de memoria se cuadruplique con cada nueva generación de chips de memoria. Un terminal adicional dedicado a direccionamiento duplica el número de filas y de columnas, y por tanto el tamaño del chip de memoria crece en un factor 4.

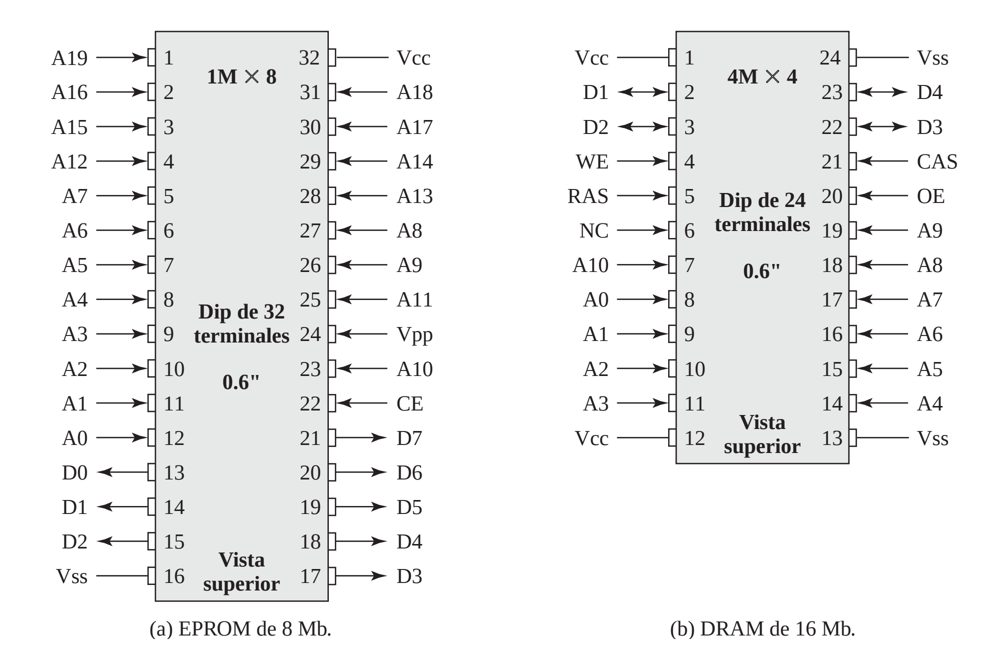
La Figura 5.3 también indica la inclusión de la circuitería de refresco. Todas las DRAM requieren operaciones de refresco. Una técnica simple de refresco consiste en inhabilitar el chip DRAM mientras se refrescan todas las celdas. El contador de refresco recorre todos los valores de fila. Para cada fila, las salidas de dicho contador se conectan al decodificador de filas y se activa la línea RAS. Los datos correspondientes se leen y escriben de nuevo en las mismas posiciones. Esto hace que se refresquen todas las celdas de una fila a la vez.

ENCAPSULADO DE LOS CHIPS

Como se mencionó en el Capítulo 2, los circuitos integrados se montan en cápsulas con patillas o terminales que los conectan con el mundo exterior.

La Figura 5.4a muestra un ejemplo de EPROM encapsulada, un chip de ocho Mb organizados en 1Mx8. En este caso, la estructura es de una-palabra-por-chip. El chip encapsulado tiene 32 terminales, siendo este uno de los tamaños estándar de encapsulado. Los terminales transfieren las siguientes señales:

- La dirección de la palabra a la que se accede. Para 1 M palabras, se necesita un total de veinte ($2^{20} = 1$M) terminales (A0-A19).
- El dato a leer, con ocho líneas (D0-D7).



(a) EPROM de 8 Mb.

(b) DRAM de 16 Mb.

Figura 5.4. Terminales y señales típicas de un chip encapsulado de memoria.

- La línea de alimentación del chip (V_{cc}).
- Un terminal de tierra (V_{ss}).
- Un terminal de habilitación de chip (CE, *Chip Enable*). Ya que puede haber varios chips de memoria, todos conectados al mismo bus de direcciones, el terminal CE se utiliza para indicar si la dirección es o no válida para cada chip. El terminal CE se activa mediante lógica cuya entrada son los bits de orden más alto del bus de direcciones (es decir, bits de direcciones superiores al A19).
- Una tensión de programación (V_{pp}) que se aplica durante la programación de la memoria (operaciones de escritura).

La Figura 5.4b muestra la configuración de terminales de un chip DRAM típico de 16 Mb organizado en 4M×4. Hay varias diferencias respecto de un chip de ROM. Ya que una RAM puede ser actualizada, los terminales de datos son de entrada/salida. Los terminales de habilitación de escritura (WE, *Write Enable*) y de habilitación de salida (OE, *Output Enable*) indican si se trata de una operación de escritura o de lectura. Debido al acceso por filas y columnas de la DRAM, y a que las direcciones están multiplexadas, solo se necesitan once terminales para especificar las cuatro megacombinaciones fila/columna (2^{11} ($2^{11} = 2^{22} = 4$M). La función de los terminales de selección de dirección de fila (RAS) y de selección de dirección de columna (CAS) ha sido descrita con anterioridad. Finalmente, el terminal «no-conectar» (NC) está simplemente para que un número total de terminales sea par.

ORGANIZACIÓN EN MÓDULOS

Si un chip de RAM contiene un bit por palabra, claramente se necesitarán al menos un número de chips igual al número de bits por palabra. Como ejemplo, la Figura 5.5 muestra cómo podría organizarse un módulo de memoria de 256 Kpalabras de ocho bits. Para 256 Kpalabras se necesitan 18 bits que se suministran al módulo desde alguna fuente externa (por ejemplo las líneas de direcciones de un bus al que esté conectado al módulo). La dirección se presenta a ocho chips de 256K (un bit, cada uno de los cuales proporciona la entrada/salida de un bit.

Esta estructura funciona cuando el tamaño de memoria sea igual al número de bits por chip. En caso de necesitar una memoria mayor, se requiere utilizar una matriz de chips. La Figura 5.6 muestra la posible organización de una memoria de 1M palabra de ocho bits. En este caso, tenemos cuatro columnas de chips, donde cada columna contiene 256 K palabras dispuestas como en la Figura 5.5. Para 1M por palabra se necesitan veinte líneas de direcciones. Los 18 bits menos significativos se conectan a los 32 módulos. Los 2 bits de orden más alto son entradas a un módulo lógico de selección de grupo que envía una señal de habilitación de chip a una de las cuatro columnas de módulos.

5.2. CORRECCIÓN DE ERRORES

Una memoria semiconductora está sujeta a errores. Estos pueden clasificarse en fallos permanentes (*hard*) y errores transitorios u ocasionales (*soft*). Un **fallo permanente** corresponde a un defecto físico, de tal modo que la celda o celdas de memoria afectadas no pueden almacenar datos de manera

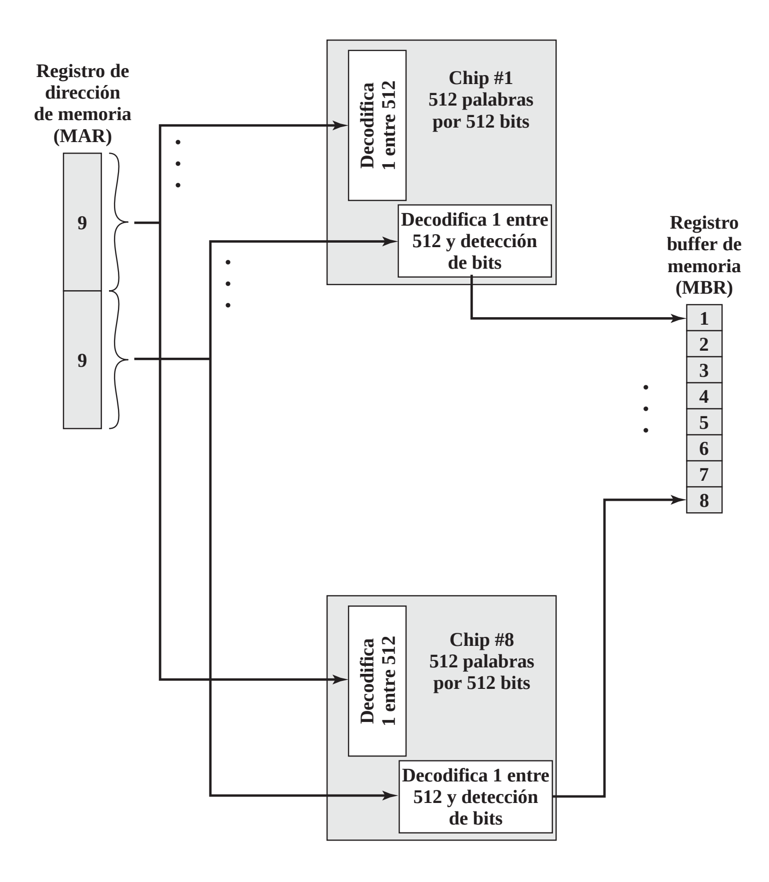


Figura 5.5. Organización de una memoria de 256 KB.

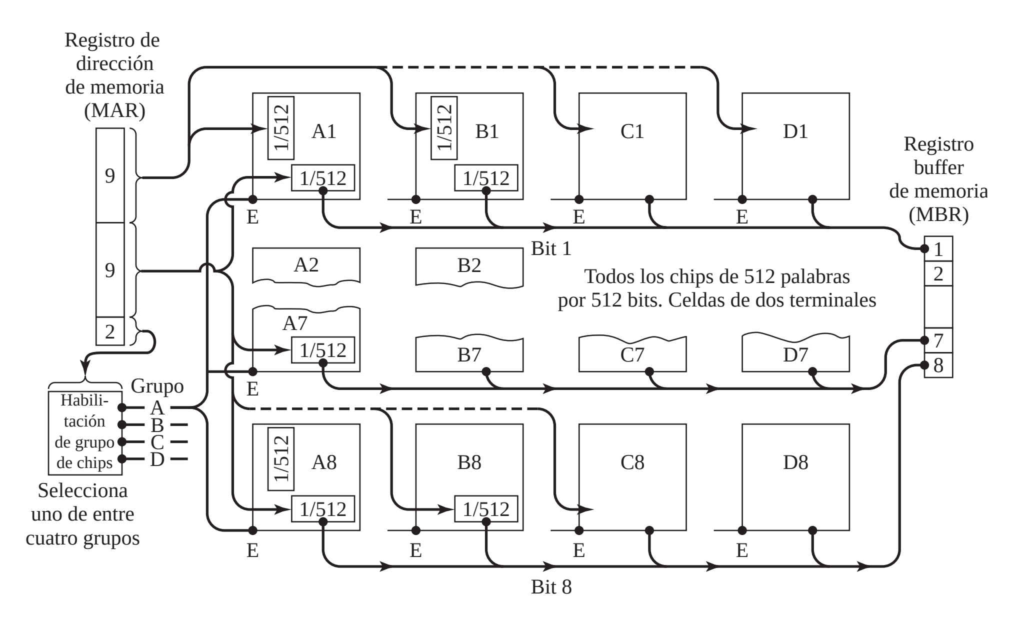


Figura 5.6. Organización de una memoria de1 MB.

segura, quedándose ancladas a cero o a uno, o conmutando erróneamente entre cero y uno. Los errores permanentes pueden estar causados por funcionamiento en condiciones adversas, defectos de fabricación, y desgaste. Un **error transitorio** es un evento aleatorio no destructivo que altera el contenido de una o más celdas de almacenamiento, sin dañar a la memoria. Los errores transitorios pueden deberse a problemas de la fuente de alimentación o a partículas alfa. Estas partículas provienen de emisión radiactiva y son lamentablemente frecuentes debido a que, en pequeñas cantidades, hay núcleos radiactivos en casi todos los materiales. Obviamente, los errores, tanto permanentes como transitorios, no son nada deseables, y la mayoría de los sistemas de memoria modernos incluyen lógica para detectar y corregir errores.

La Figura 5.7 ilustra, en términos generales, cómo se lleva a cabo el proceso. Cuando se van a escribir datos en memoria, se realiza un cálculo con los datos, representado por la función *f*, para producir un código. Se almacenan tanto los datos como el código. Así, si se va a almacenar una palabra de datos de *M* bits, y el código tiene una longitud de *K* bits, el tamaño real de la palabra almacenada es de *M*+*K* bits.

Cuando se va a leer una palabra previamente almacenada, se utiliza el código para detectar errores, y puede que incluso corregirlos. Se genera un nuevo código de *K* bits a partir de los *M* bits de datos, que se compara con los bits de código captados de memoria. Esta comparación produce uno de tres resultados posibles:

- No se detectan errores. Los bits de datos captados se envían al exterior.
- Se detecta un error y es posible corregirlo. Se dan a un corrector los bits de datos más los bits de corrección de error, lo que produce un conjunto corregido de *M* bits a ser enviados fuera.
- Se detecta un error, pero no es posible corregirlo. Se informa de esta situación.

Los códigos que operan de esta manera se denominan *códigos correctores de errores*. Un código se caracteriza por el número de bits de error de una palabra que puede corregir y detectar.

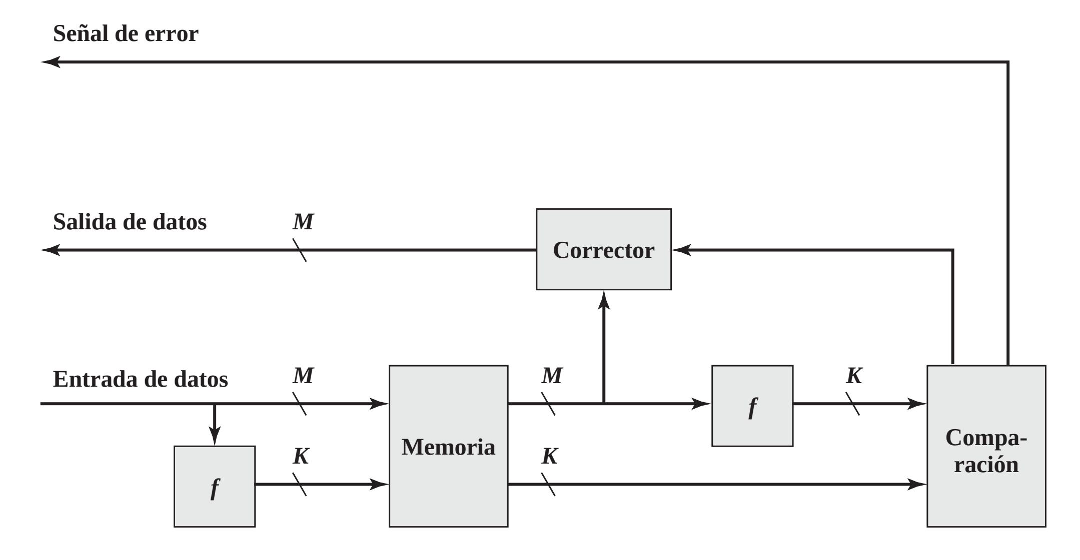


Figura 5.7. Función del código de correciones de errores.

El código corrector de errores más sencillo es el *código de Hamming*, ideado por Richard Hamming en los Laboratorios Bell. La Figura 5.8 ilustra el uso de este código mediante diagramas de Venn, con palabras de cuatro bits ($M = 4$). Al intersectarse tres círculos se tienen siete compartimentos. Asignamos los cuatro bits del dato a los compartimentos interiores (Figura 5.8a). Los restantes compartimentos se rellenan con los denominados *bits de paridad*. Cada bit de paridad se elige de tal manera que el número total de unos en su círculo sea par (Figura 5.8b). Así pues, ya que el círculo A incluye tres unos del dato, el bit de paridad se pone a uno en dicho círculo. Ahora, si un error cambia uno de los bits de datos (Figura 5.8c), se encuentra fácilmente. Comprobando los bits de paridad, se encuentran discrepancias en los círculos A y C pero no en B. Solo uno de los siete compartimentos está en A y en C pero no en B. El error puede pues corregirse modificando el bit de dicho compartimento.

Para clarificar ideas, desarrollaremos un código que puede detectar y corregir errores de un solo bit en palabras de ocho bits.

Para empezar determinemos cuán largo debe ser el código. Con referencia a la Figura 5.7, la lógica de comparación recibe como entrada dos valores de K bits. La comparación bit a bit se hace mediante la OR-exclusiva de las dos entradas. El resultado se denomina *palabra de síndrome*. Cada bit del síndrome es 0 o 1 según que haya o no coincidencia en esa posición de bit para las dos entradas.

La palabra de síndrome tiene pues una longitud de K bits y tiene un rango entre 0 y $2^K - 1$. El valor 0 indica que no se ha detectado error, dejando $2^K - 1$ valores para indicar, si hay error, qué bit fue el erróneo. Ahora, ya que el error podría haber ocurrido en cualquiera de los M bits de datos o de los K bits de comprobación, se debe cumplir:

$$2^k - 1 \geq M + K$$

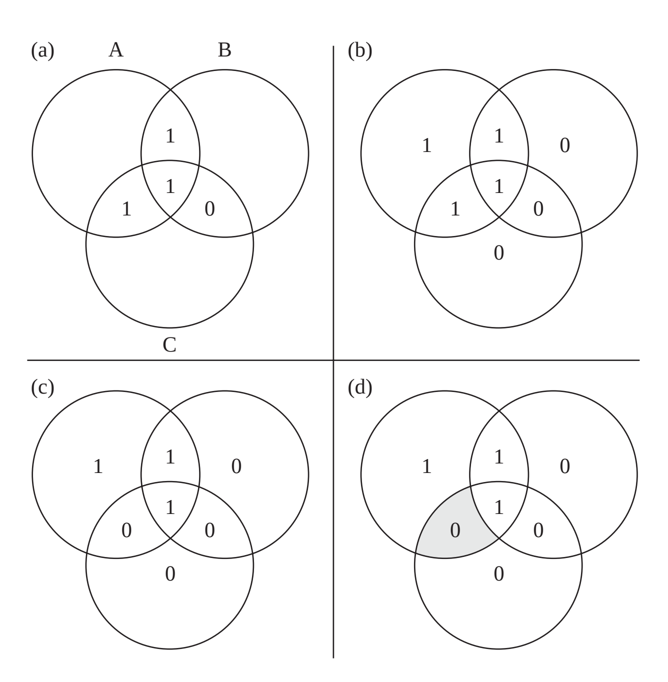


Figura 5.8. Código corrector de errores de Hamming.

Esta inecuación da el número de bits necesarios para corregir el error, de un solo bit cualquiera, en una palabra que contenga *M* bits de datos. Por ejemplo, para una palabra de ocho bits de datos ($M = 8$) tenemos:

- $K = 3: 2^3 - 1 < 8 + 3$
- $K = 4: 2^4 - 1 > 8 + 4$

Por lo tanto, ocho bits de datos requieren de cuatro bits de comprobación. Las tres primeras columnas de la Tabla 5.2 listan los números de bits de comprobación necesarios para distintas longitudes de palabra.

Por conveniencia, para una palabra de datos de ocho bits sería deseable generar un síndrome de cuatro bits con las siguientes características:

- Si el síndrome contiene solo ceros, no se ha detectado error.
- Si el síndrome contiene solo un bit puesto a 1, ha ocurrido un error en uno de los cuatro bits de comprobación. No se requiere corrección.
- Si el síndrome contiene más de un bit puesto a 1, entonces el valor numérico de dicho síndrome indica la posición del bit de dato erróneo. Se invierte dicho bit de dato para corregirlo.

Para conseguir estas características, los bits de datos y de comprobación se distribuyen en una palabra de doce bits como se muestra en la Figura 5.9. Las posiciones de bit están numeradas de uno

Tabla 5.2. Aumento de la longitud de palabra con la corrección de errores.

	Correción de errores simples		Corrección de errores simples/ detección de errores dobles	
Bits de datos	**Bits de comprobación**	**% incremento**	**Bits de comprobación**	**% incremento**
8	4	50	5	62,5
16	5	31,25	6	37,5
32	6	18,75	7	21,875
64	7	10,94	8	12,5
128	8	6,25	9	7,03
256	9	3,52	10	3,91

Posición de bit	12	11	10	9	8	7	6	5	4	3	2	1
Número de posición	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Bit de datos	D8	D7	D6	D5		D4	D3	D2		D1		
Bit de comprobación					C8				C4		C2	C1

Figura 5.9. Posiciones de los bits de datos y de comprobación.

a doce. A los bits de comprobación se asignan aquellas posiciones de bit cuyos números son potencias de dos. Los bits de comprobación se calculan como sigue:

$$
\begin{aligned}
C1 &= D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7 \\
C2 &= D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7 \\
C4 &= D2 \oplus D3 \oplus D4 \oplus D8 \\
C8 &= D5 \oplus D6 \oplus D7 \oplus D8
\end{aligned}
$$

donde los símbolos $\oplus$ designan la operación XOR (OR-exclusiva).

Cada bit de comprobación opera sobre todo bit de datos cuyo número de posición contiene un 1 en la misma posición que el número de posición del bit de comprobación. Así pues, las posiciones de bit de datos 3, 5, 7, 9, y 11 (D1, D2, D4, D5, D7) tienen un 1 en el bit menos significativo de su numero de posición, igual que ocurre con C1; las posiciones 3, 6, 7, 10 y 11 tienen un 1 en la segunda posición de bit, lo mismo que C2; y así sucesivamente. Visto de otra forma, la posición de bit n es comprobada por aquellas posiciones de bit C_i tal que $\Sigma i = n$. Por ejemplo, la posición 7 es comprobada por los bits en las posiciones 4, 2 y 1; y 7 = 4 + 2 + 1.

Verifiquemos con un ejemplo que el esquema anterior funciona. Supongamos que la palabra de entrada de ocho bits es 00111001, siendo el bit de dato D1 el de la posición más a la derecha. Los cálculos son los siguientes:

$$
\begin{aligned}
C1 &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\
C2 &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\
C4 &= 0 \oplus 0 \oplus 1 \oplus 0 = 1 \\
C8 &= 1 \oplus 1 \oplus 0 \oplus 0 = 0
\end{aligned}
$$

Supongamos ahora que el bit de datos 3 se ve afectado por un error, cambiando de 0 a 1. Cuando se recalculan los bits de comprobación, se tiene:

$$
\begin{aligned}
C1 &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1 \\
C2 &= 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 1 \\
C4 &= 0 \oplus 1 \oplus 1 \oplus 0 = 0 \\
C8 &= 1 \oplus 1 \oplus 0 \oplus 0 = 0
\end{aligned}
$$

Cuando se comparan los nuevos bits de comprobación con los antiguos, se genera la palabra de síndrome:

$$
\begin{array}{ccccc}
 & C8 & C4 & C2 & C1 \\
 & 0 & 1 & 1 & 1 \\
\oplus & 0 & 0 & 0 & 1 \\
\hline
 & 0 & 1 & 1 & 0
\end{array}
$$

El resultado es 0110, indicando que la posición de bit 6, que contiene el bit 3 del dato, es errónea.

La Figura 5.10 ilustra el cálculo anterior. Los bits de datos y de comprobación se ubican convenientemente en la palabra de doce bits. Cuatro de los bits de datos valen 1 (sombreados en la tabla),

Posición de bit	12	11	10	9	8	7	6	5	4	3	2	1
Número de posición	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Bit de datos	D8	D7	D6	D5		D4	D3	D2		D1		
Bit de comprobación					C8				C4		C2	C1
Palabra memorizada como:	0	0	1	1	0	1	0	0	1	1	1	1
Palabra captada como:	0	0	1	1	0	1	1	0	1	1	1	1
Número de posición	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Bit de comprobación					0				0		0	1

Figura 5.10. Cálculo de los bits de comprobación.

y la operación XOR de sus valores de posición de bit produce el código Hamming 0111, que se corresponde con los cuatro dígitos de comprobación. El bloque entero que se almacena es 001101001111. Suponga ahora que el bit 3 de datos, en la posición de bit 6, tiene un error y cambia de 0 a 1. Una XOR del código Hamming con todos los valores de posición de bit para los que los bits de datos son distintos de cero, da como resultado: 0110. Este resultado indica que se ha detectado un error y que el bit erróneo está en la posición 6.

El código que acabamos de describir es conocido como código *corrector de errores simples* (SEC). Es más común equipar las memorias semiconductoras con un código *corrector de errores simples y detector de errores dobles* (SEC-DED). Como muestra la Tabla 5.2, estos códigos necesitan un bit más que los SEC.

La Figura 5.11 ilustra, para una palabra de datos de cuatro bits, cómo funciona un código SEC-DED. La secuencia de la figura muestra que si ocurren dos errores (Figura 5.11c), el procedimiento

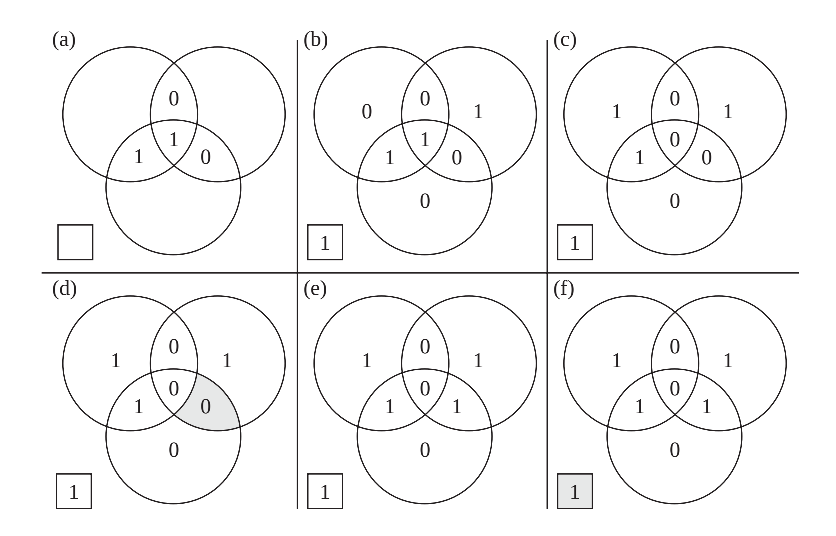


Figura 5.11. Código SEC-DED de Hamming.

de chequeo que conocíamos informa erróneamente (d) y empeora el problema creando un tercer error (e). Para superar el problema, se añade un octavo bit cuyo valor se fija de manera que el número total de unos en el diagrama sea par. El bit de paridad añadido captura el error (f).

Un código corrector de errores mejora la fiabilidad de la memoria a costa de una complejidad adicional. Con una organización de un-bit-por-chip, es generalmente adecuado considerar un código SEC-DED. Por ejemplo, los IBM 30xx utilizaban un código SEC-DED de ocho bits por cada 64 bits de datos en memoria principal. Así, el tamaño de la memoria principal es realmente un 12 por ciento mayor que el aparente para el usuario. Los computadores VAX utilizaron un código SEC-DED de 7 bits por cada 32 bits de memoria; un 22 por ciento de incremento. Diversas memorias DRAM modernas emplean 9 bits de comprobación por cada 128 bits de datos; un 7 por ciento de incremento [SHAR97].

5.3. ORGANIZACIÓN AVANZADA DE MEMORIAS DRAM

Como se discutió en el Capítulo 2, uno de los cuellos de botella más críticos de un sistema que utiliza procesadores de altas prestaciones es la interfaz con la memoria principal interna. Esta interfaz es el camino más importante en el computador. El bloque básico de construcción de la memoria principal sigue siendo el chip de DRAM, como lo ha sido durante décadas, y desde principios de la década de los 1970 no ha habido cambios significativos en la arquitectura DRAM. El chip DRAM tradicional está limitado tanto por su arquitectura interna como por su interfaz con el bus de memoria del procesador.

Hemos visto que una forma de abordar el problema de las prestaciones de la memoria principal DRAM ha sido insertar uno o más niveles de cachés SRAM de alta velocidad entre la memoria principal DRAM y el procesador. Pero la SRAM es mucho más costosa que la DRAM, y ampliar el tamaño de cachés más allá de cierta cantidad produce menos beneficios.

En los últimos años se han explorado diversas versiones mejoradas de la arquitectura básica DRAM, y algunas de ellas están siendo comercializadas. Los esquemas que dominan actualmente el mercado son: SDRAM, DDR-DRAM, y RDRAM. La Tabla 5.3 proporciona una comparativa de sus prestaciones. Las CDRAM han sido también motivo de atención. Esta sección da una visión de estas nuevas tecnologías de DRAM.

DRAM SÍNCRONA

Una de las formas de DRAM más ampliamente usadas es la DRAM síncrona (SDRAM) [VOGL94]. A diferencia de las DRAM tradicionales, que son asíncronas, la SDRAM intercambia datos con el

Tabla 5.3. Comparación de prestaciones de diversas DRAM.

	Frecuencia de reloj (MHz)	Velocidad de transferencia (GB/s)	Tiempo de acceso (ns)	Número de terminales
SDRAM	166	1.6	18	168
DDR	200	3.2	12.5	184
RDRAM	600	4.8	12	162

procesador de forma sincronizada con una señal de reloj externa, funcionando a la velocidad tope del bus procesador/memoria, sin imponer estados de espera.

En una DRAM típica, el procesador presenta las direcciones y niveles de control a la memoria, indicando que los datos de una posición de memoria concreta deben bien escribirse o leerse. Después de un tiempo, el tiempo de acceso, se escriben o leen los datos en la DRAM. Durante el tiempo de acceso, la DRAM realiza varias operaciones internas, tales como activar las capacidades elevadas de las líneas de fila y de columna, detectar los datos, y sacarlos a través de los buffers de salida. El procesador debe simplemente esperar durante este tiempo, haciendo que el sistema baje en prestaciones.

Con el acceso síncrono, la DRAM introduce y saca datos bajo el control del reloj del sistema. El procesador, u otro maestro, cursa la información de instrucción y de dirección, que es retenida por la DRAM. La DRAM responderá después de un cierto número de ciclos de reloj. Entre tanto, el maestro puede realizar sin riesgo otras tareas mientras la SDRAM esta procesando la petición.

La Figura 5.12 muestra la lógica interna de una SDRAM de 64 Mb de IBM [IBM01], que es una estructura típica de SDRAM, y la Tabla 5.4 define la asignación de sus terminales.

La SDRAM emplea un modo de ráfagas para eliminar los tiempos de establecimiento de dirección y de precarga de las líneas de fila y de columna posteriores al primer acceso. En el modo de ráfagas, se puede secuenciar la salida rápida de una serie de bits de datos una vez que se ha accedido al

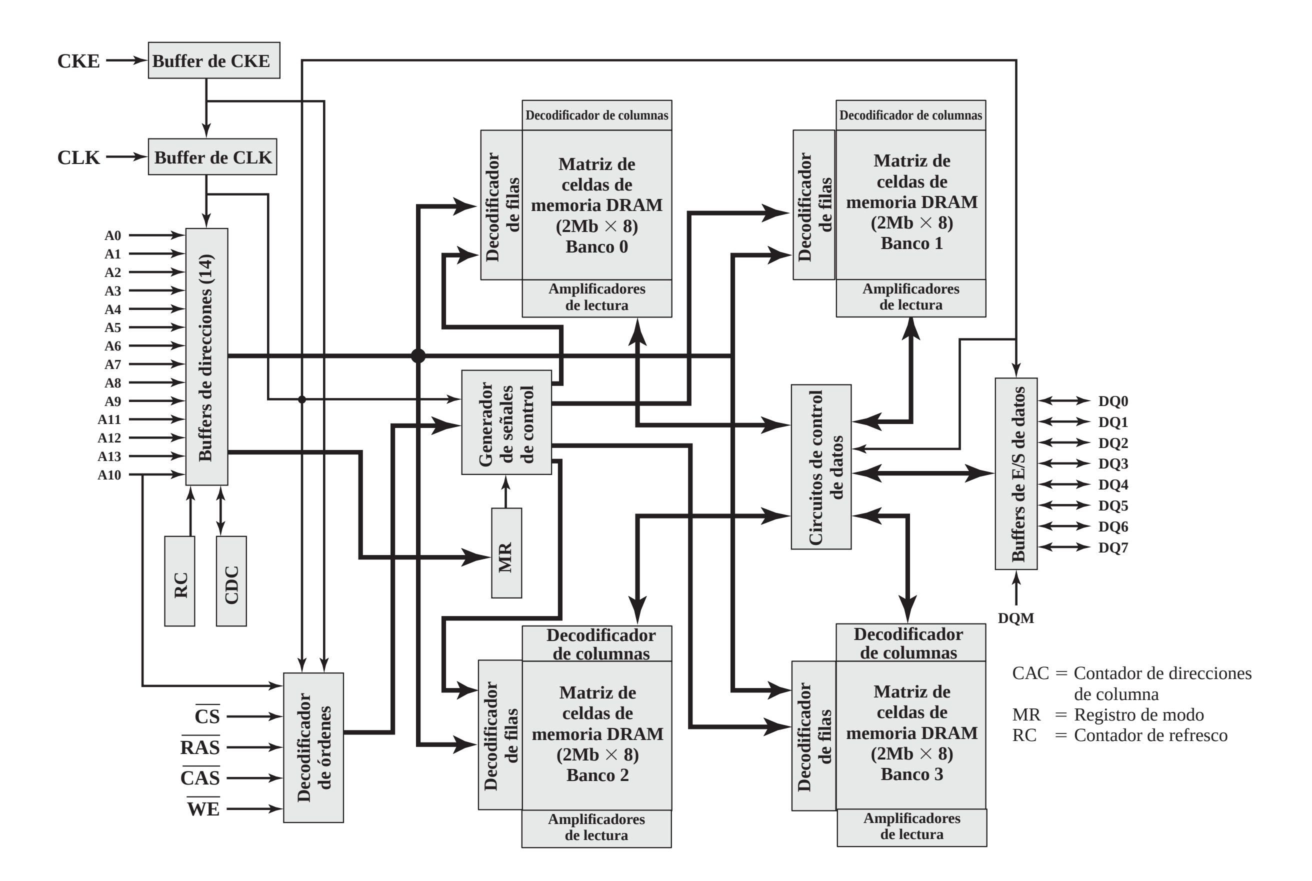


Figura 5.12. RAM dinámica síncrona (SDRAM).

Tabla 5.4. Asignaciones de terminales de la SDRAM.

A0 a A13	Entradas de direcciones
CLK	Entrada de reloj
CKE	Habilitación del reloj
$\overline{\text{CS}}$	Selección de chip
$\overline{\text{RAS}}$	Selección de dirección de fila
$\overline{\text{CAS}}$	Selección de dirección de columna
$\overline{\text{WE}}$	Habilitación de escritura
DQ0 a DQ7	Entradas/salidas de datos
DQM	Máscara de datos

primero de ellos. Este modo es útil cuando todos los bits a acceder están en secuencia y en la misma fila de la matriz de celdas que el accedido en primer lugar. Además, la SDRAM tiene una arquitectura interna de banco múltiple que facilita el paralelismo en el propio chip.

El registro de modo y la lógica de control asociada constituyen otra característica clave que diferencia las SDRAM de la DRAM convencionales. Proporciona una manera de particularizar la SDRAM para ajustarse a las necesidades concretas del sistema. El registro de modo especifica la longitud de la ráfaga, que es el número de unidades individuales de datos que se entregan síncronamente al bus. Este registro también permite al programador ajustar la latencia entre la recepción de una petición de lectura y el comienzo de la transferencia de datos.

La SDRAM funciona mejor cuando transfiere bloques largos de datos en serie, tal como en aplicaciones de procesamiento de textos, hoja de cálculo y multimedia.

La Figura 5.13 muestra un ejemplo de funcionamiento de una SDRAM. En este caso, La longitud de ráfaga vale 4 y la latencia es 2. La orden de lectura en ráfaga se inicia teniendo $\overline{CS}$ y $\overline{CAS}$ en baja mientras se mantienen $\overline{RAS}$ y $\overline{WE}$ en alta al llegar el flanco ascendente del reloj. Las entradas de direcciones determinan la dirección de columna inicial para la ráfaga, y el registro de modo indica el tipo de ráfaga (secuencial o entrelazada) y la longitud de la ráfaga (1, 2, 4, 8, página completa). El retardo desde el inicio de la orden hasta que el dato de la primera celda aparece en las salidas coincide con el valor de latencia de que se ha fijado en registro de modo.

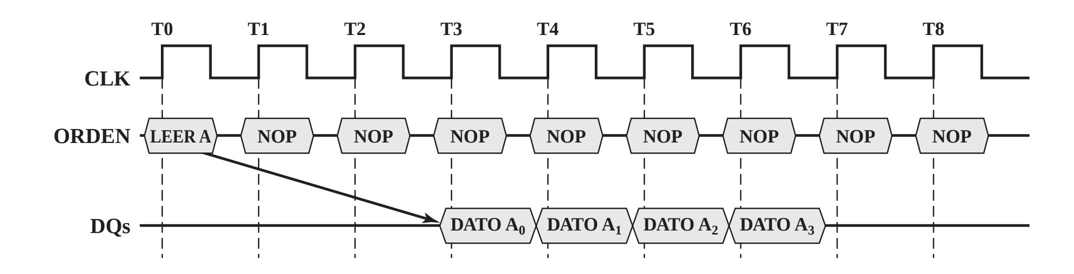


Figura 5.13. Temporización de una lectura de SDRAM (longitud de ráfaga = 4, latencia de $\overline{\text{CAS}}$ = 2).

Ahora existe una versión mejorada de SDRAM, conocida como SDRAM de doble velocidad de datos (DDR-SDRAM, *double data rate SDRAM*) que supera la limitación de uno-por-ciclo. Una DDR-SDRAM puede enviar datos al procesador dos veces por ciclo.

DRAM RAMBUS

La RDRAM, desarrollada por Rambus [FARM92, CRIS97], ha sido adoptada por Intel para sus procesadores Pentium e Itanium. Se ha convertido en la principal competidora de la SDRAM. Los chips RDRAM tienen encapsulados verticales, con todos los terminales en un lateral. El chip intercambia datos con el procesador por medio de 28 hilos de menos de doce centímetros de longitud. El bus puede direccionar hasta 320 chips de RDRAM y a razón de 1,6 GBps.

El bus especial de las RDRAM entrega direcciones e información de control utilizando un protocolo asíncrono orientado a bloques. Tras un tiempo de acceso inicial de 480 ns, se consigue la velocidad de datos de 1,6 GBps. Lo que hace posible esta velocidad es el bus en sí, que define muy precisamente las impedancias, la temporización, y las señales. En lugar de ser controladas por las señales explícitas RAS, CAS, R/W, y CE que se utilizan en DRAM convencionales, las RDRAM obtienen las peticiones de memoria a través de un bus de alta velocidad. Cada petición contiene la dirección deseada, el tipo de operación, y el número de bytes en dicha operación.

La Figura 5.14 muestra el esquema de RDRAM. La configuración consta de un controlador y de varios módulos RDRAM conectados juntos mediante un bus común. El controlador está en un extremo de la configuración, y el extremo más alejado del mismo es un terminador paralelo de las líneas del bus. El bus incluye 18 líneas de datos (realmente 16, más dos de paridad) que circulan al doble de la velocidad del reloj; es decir se envía un bit por cada uno de los dos flancos de un ciclo de reloj. Esto hace que la velocidad de transferencia en cada línea de datos sea de 800 Mbps. Existe un conjunto aparte de ocho líneas (RC) que se emplea para direcciones y señales de control. Hay también una señal de reloj que parte del extremo más alejado del controlador, se propaga hacia él y después retorna. Un módulo RDRAM envía datos al controlador en sincronismo con el reloj directo, y el controlador envía datos a una RDRAM en sincronismo con la señal de reloj en sentido opuesto. Las restantes líneas del bus incluyen una tensión de referencia, tierra, y la tensión de alimentación.

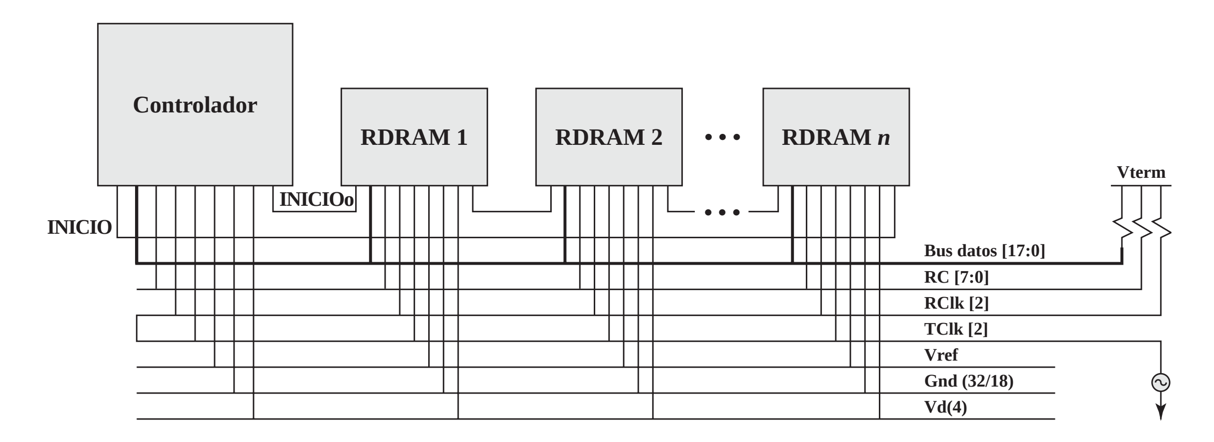


Figura 5.14. Estructura RDRAM.

DDR SDRAM

La SDRAM está limitada por el hecho de que puede enviar datos al procesador solo una vez por ciclo de reloj del bus. Una nueva versión de SDRAM, denominada SDRAM de doble velocidad de datos (DDR-SDRAM), puede enviar datos dos veces cada ciclo de reloj, una coincidiendo con el flanco de subida del pulso de reloj y otra coincidiendo con el flanco de bajada.

DRAM CACHÉS

La DRAM Cachés (CDRAM), desarrollada por Mitsubishi [HIDA90, ZHAN01], integra una pequeña caché SRAM (de 16 Kb) en un chip normal de DRAM.

La SRAM de la CDRAM puede usarse de dos formas. En primer lugar, puede utilizarse como una verdadera caché, formada por líneas de 64 bits. El modo cachés de la CDRAM es efectivo para accesos a memoria aleatorios ordinarios.

La SRAM de la CDRAM puede usarse también como buffer para soportar el acceso serie a un bloque de datos. Por ejemplo, para refrescar una pantalla gráfica, la CDRAM puede precaptar en la SRAM los datos de la DRAM, de manera que los accesos posteriores al chip se efectúen únicamente a la SRAM.

5.4. LECTURAS Y SITIOS WEB RECOMENDADOS

[PRIN97] proporciona un tratamiento amplio de las tecnologías de memorias semiconductoras, incluyendo SRAM, DRAM, y memorias flash. [SHAR97] cubre también los mismos temas, haciendo más hincapié en aspectos relativos a test y fiabilidad. [SHAR03] y [PRIN96] se centran en arquitecturas avanzadas de DRAM y SRAM. Para una revisión en profundidad de las DRAM, *véase* [KEET01]. [CUPP01] proporciona una interesante comparación de prestaciones de distintos esquemas DRAM. [BEZ03] introduce con detalle la tecnología de memorias flash.

[MCEL85] contiene una buena explicación de los códigos de corrección de errores. Para un estudio más profundo merecen la pena los libros [ADAM91] y [BLAH83]. En [ASH90] se da un tratamiento teórico y matemático, fácilmente legible, de los códigos de corrección de errores. [SHAR97] contiene una buena revisión de los códigos utilizados en memorias modernas.

ADAM91 Adamek, J.: *Foundations of Coding.* New York. Wiley, 1991.

ASH90 Ash, R.: *Information Theory.* New York. Dover, 1990.

BEZ03 Bez, R., *et al.*: Intoduction to Flash Memory. *Proceedings of the IEEE.* Abril, 2003.

BLAH83 Blahut, R.: *Theory and Practice of Error Control Codes.* Reading, MA. Addison-Wesley, 1983.

CUPP01 Cuppu, V., *et al.*: «Higt Performance DRAMS in Workstation Environments». *IEEE Transactions on Computers,* noviembre, 2001.

KEET01 Keeth, B. y Baker, R.: *DRAM Circuit Desing: A Tutorial. Piscataway,* NJ. IEEE Press, 2001.

MCEL85 McEliece, R.: «The Reliability of Computer Memories». *Scientific American,* enero, 1985.

PRIN97 Prince, B.: *Semiconductor Memories.* New York. Wiley, 1997.

PRIN 02 Prince, B. *Emerging Memories: Technologies and Trends.* Norwell, MA. Kluer, 2002.

SHAR97 Sharma, A.: *Semiconductor Memories: Technology, Testing, and Reliability.* New York. IEEE Press, 1997.

SHAR03 Sharma, A.: *Advanced Semiconductor Memories: Architectures, Desings, and Applications.* New York. IEEE Press, 2003.



SITIOS WEB RECOMENDADOS

- **The RAM Guide:** una buena revisión sobre tecnología RAM y múltiples enlaces de utilidad.
- **RDRAM:** otro sitio útil con información de RDRAM

5.5. PALABRAS CLAVE, CUESTIONES DE REPASO Y PROBLEMAS

PALABRAS CLAVE

código corrector de errores simples (SEC) código corrector de errores simples, detector de errores dobles (SEC-DEC)	DRAM síncrona (SDRAM)	memoria volátil
código de corrección de errores (ECC)	error transitorio	RAM dinámica (DRAM)
corrección de errores	fallo permanente	RAM estática (SRAM)
DRAM cachés (CDRAM)	memoria flash código de Hamming	ROM borrable y programable eléctricamente (EEPROM)
DRAM RamBus (RDRAM)	memoria de sobre-todo lectura	ROM programable (PROM)
	memoria de solo lectura (ROM)	ROM programable y borrable (EPROM)
	memoria no volátil	síndrome
	memoria semiconductora	

PREGUNTAS DE REPASO

5.1. ¿Cuáles son las propiedades clave de las memorias semiconductoras?

5.2. ¿Cuál de los dos significados se está empleando para el término *memoria de acceso aleatorio?*

5.3. ¿Qué diferencia hay, en cuanto a aplicaciones, entre DRAM y SRAM?

5.4. ¿Qué diferencia hay entre DRAM y SRAM en cuanto a características tales como velocidad, tamaño y coste?

5.5. Explique por qué uno de los tipos de RAM se considera analógico y el otro digital.

5.6. Indique algunas aplicaciones de las ROM.

5.7. ¿Qué diferencias hay entre las memorias EPROM, EEPROM y flash?

5.8. Explique la función de cada uno de los terminales de la Figura 4.5b.

5.9. ¿Qué es un bit de paridad?

5.10. ¿Cómo se interpreta el síndrome en el código Hamming?

5.11. ¿Qué diferencia hay entre una SDRAM y una DRAM convencional?

PROBLEMAS

5.1. Sugiera razones por las que las RAM han sido tradicionalmente organizadas en solo un bit por chip mientras que las ROM están normalmente organizadas en múltiples bits por chip.

5.2. Considere una RAM dinámica a la que deba darse un ciclo de refresco 64 veces por milisegundo. Cada operación de refresco requiere 150 ns; un ciclo de memoria requiere 250 ns. ¿Qué porcentaje del tiempo total de funcionamiento de la memoria debe dedicarse a los refrescos?

5.3. La Figura 5.15 muestra un diagrama de tiempos simplificado del bus durante una operación de lectura de DRAM. El tiempo de acceso se considera que es desde t_1 hasta t_2. A continuación hay un tiempo de recarga, que dura desde t_2 hasta t_3, durante el cual los chips de DRAM tienen que recargar antes de que el procesador puede accederlos de nuevo.

(a) Suponga que el tiempo de acceso es de 60 ns y que el tiempo de recarga es de 40 ns. ¿Qué valor tiene el tiempo de ciclo de memoria? ¿Qué velocidad máxima de transferencia de datos puede mantener esta memoria, suponiendo que su salida es de 1 bit?

(b) Si se construye un sistema de memoria para datos de 32 bits usando estos chips, ¿qué transferencia de datos se obtiene?

5.4. La Figura 5.6 muestra cómo construir un módulo de chips que pueden memorizar 1 Mb a partir de un grupo de cuatro chips de 256 Kb. Supongamos que este módulo de varios chips estuviera encapsulado como un único chip de 1 Mb, con tamaño de palabra de 1 byte. Dibuje un diagrama que especifique cómo construir una memoria de 8 Mb utilizando ocho de estos chips de 1 Mb. Asegúrese de indicar en el diagrama cómo se utilizan las distintas líneas de direcciones.

5.5. En un sistema típico basado en un Intel 8086, conectado a DRAM a través del bus del sistema, para una operación de lectura se activa $\overline{RAS}$ mediante el flanco siguiente de la señal de Habilitación de Direcciones (*address enable,* Figura 3.19). Sin embargo, debido al tiempo de propagación y a otras componentes de retardo, $\overline{RAS}$ no se activa hasta 50 ns después de que la línea Habilitación de Direcciones haya vuelto a baja. Suponga que esto ocurre en medio de la segunda mitad del estado T_2 (un poco antes que en la Figura 3.19). El procesador lee los datos al final de T_3, pero conviene que la memoria los presente con 60 ns de antelación. Este intervalo de tiempo tiene en cuenta los retardos de propagación a través del camino de datos (desde memoria hasta el procesador) y el tiempo necesario de retención de datos. Suponga una frecuencia de reloj de 10 MHz.

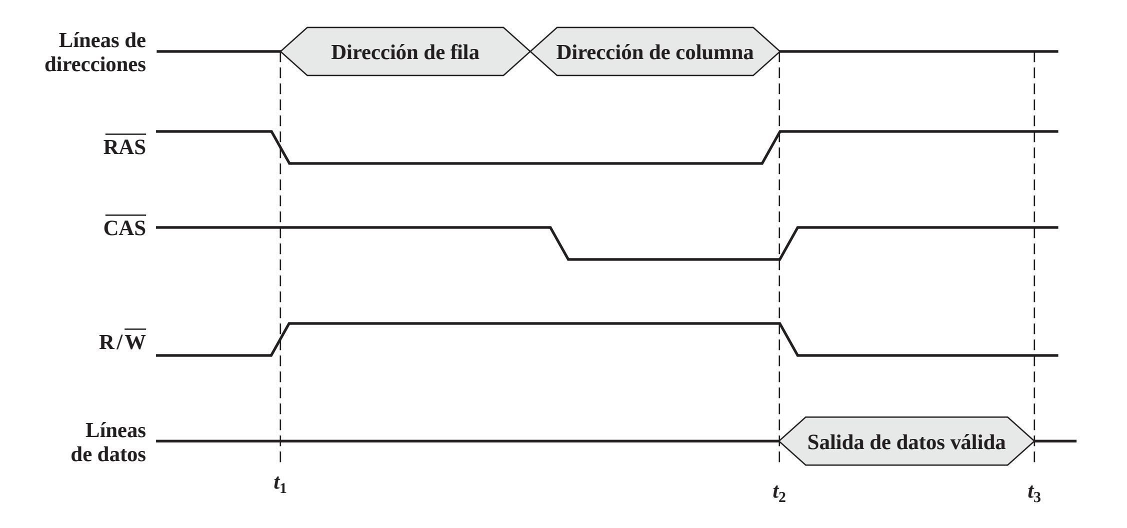


Figura 5.15. Diagrama de tiempos simplificado de una lectura de DRAM.

(a) ¿Cómo de rápidas (tiempo de acceso) debieran ser las DRAM para que no se inserten estados de espera?

(b) ¿Cuántos estados de espera se deben insertar por cada operación de lectura de memoria si el tiempo de acceso de las DRAM es de 150 ns?

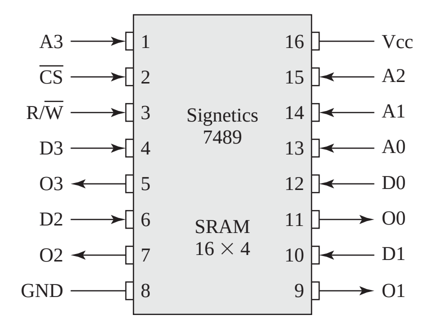
5.6. La memoria de un cierto microcomputador está construida a partir de chips DRAM de 64K × 1. De acuerdo con su hoja de características, la matriz de celdas del chip de RAM está organizada en 256 filas. Cada fila debe refrescarse al menos una vez cada 4 ms. Suponga que refrescamos la memoria con una periodicidad estricta.

(a) ¿Qué tiempo transcurre entre peticiones de refresco sucesivas?

(b ¿Qué tamaño debe tener el contador de direcciones de refresco?

5.7. La Figura 5.16 muestra una de las primeras SRAM, el chip 7489 de Signetics, de 16 × 4, que almacena 16 palabras de 4 bits.

(a) Explique el modo de funcionamiento del chip para cada pulso de entrada $\overline{CS}$ indicado en la Figura 5.16c.

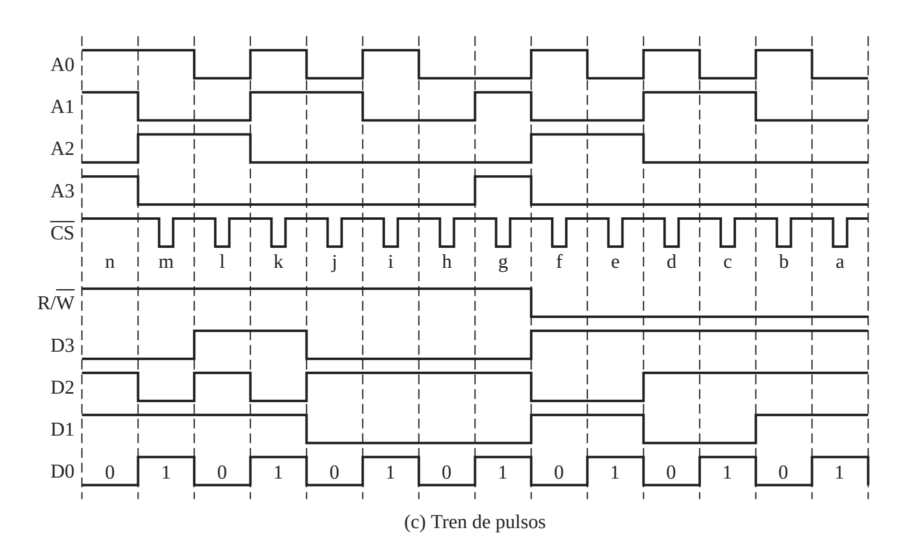


(a) Asignación de terminales

Modo de funcionamiento	Entradas			Salidas
	$\overline{CS}$	R/$\overline{W}$	D*n*	O*n*
Escritura	L	L	L	L
	L	L	H	H
Lectura	L	H	X	Dato
Inhibe escritura	H	L	L	H
	H	L	H	L
Memorizar-inhabilitar salidas	H	H	X	H

H = Nivel de voltaje alto
L = Nivel de voltaje bajo
X = Indiferente

(b) Tabla de verdad



(c) Tren de pulsos

Figura 5.16. La SRAM de Signetics 7489.

(b) Indique el contenido de la memoria en las posiciones de palabra 0 a 6 después del pulso *n*.

(c) ¿Qué valores se tienen en los datos de salida para los pulsos de entrada *m* a *h*?

5.8. Diseñe una memoria de 16-bits con una capacidad total de 8192 bits utilizando chips de SRAM de tamaño 64×1 bit. Indique la configuración matricial de los chips en la tarjeta de memoria, mostrando todas las señales de entrada y salida necesarias para asignar esta memoria al espacio de direcciones más bajo. El diseño debe permitir accesos tanto por bytes como por palabras de 16 bits.

5.9. Una unidad de medida usual para la tasa de fallos de los componentes electrónicos es la **unidad de fallos** FIT (de *Failure unIT*), expresada en número de fallos por cada mil millones de horas del dispositivo. Otra medida conocida, aunque menos usada, es el **tiempo medio entre fallos** (MTBF), que es el tiempo medio de funcionamiento de un determinado componente hasta que falla. Considere una memoria de 1 MB, para un procesador de 16 bits, implementada con chips DRAM de $256K \times 1$. Calcule su MTBF suponiendo 2000 FITS para cada chip DRAM.

5.10. Para el código de Hamming de la Figura 5.10, indique qué ocurre cuando el error se produce en uno de los bits de comprobación en lugar de un bit de datos.

5.11. Considere la siguiente palabra de ocho bits almacenada en memoria: 11000010. Utilizando el algoritmo de Hamming, determine qué bits de comprobación se memorizarían junto con la palabra de datos. Muestre cómo ha obtenido el resultado.

5.12. Para la palabra de datos de ocho bits 00111001, los bits de comprobación que se memorizan junto con ella serían: 0111. Suponga que al leer la palabra de memoria se calculan los bits de comprobación: 1101 ¿Cuál es la palabra de datos leída de memoria?

5.13. ¿Cuántos bits de comprobación se necesitan para utilizar el código de corrección de errores de Hamming en la detección de errores de un solo bit en una palabra de datos de 1024 bits?

5.14. Desarrolle un código SEC para palabras de datos de 16 bits. Genere el código para la palabra de datos 0101000000111001. Demuestre que el código identificará correctamente un error en el bit 5 de datos.

CAPÍTULO 6

Memoria externa

PUNTOS CLAVE

- Los discos magnéticos siguen siendo el componente más importante de la memoria externa. Tanto los extraíbles como los fijos, o duros, los discos se usan tanto en los PC, como en computadores grandes y supercomputadores.
- Para conseguir mayores prestaciones y disponibilidad, un esquema de servidores y sistemas grandes extendido es la tecnología RAID de discos. RAID se refiere a una familia de técnicas para utilizar varios discos como un conjunto de dispositivos de almacenamiento de datos en paralelo, con redundancia para compensar los fallos de disco.
- Las técnicas de almacenamiento óptico se han convertido en algo cada vez más importante en los computadores. Mientras que el CD-ROM se ha usado ampliamente durante muchos años, tecnologías más recientes, como el CD reescribible y las unidades de almacenamiento magnético-ópticas, están siendo cada vez más importantes.

En este capítulo se examinan distintos sistemas y dispositivos de memoria externa. Comenzamos con el dispositivo más importante, el disco magnético. Los discos magnéticos son la base de las memorias externas en casi todos los computadores. En la siguiente sección se examina el uso de conjuntos de discos para conseguir mayores prestaciones, concretamente la familia conocida como RAID (*Redundant Array of Independent Disks*, conjunto redundante de discos independientes). La memoria óptica externa es un componente cada vez más importante de muchos computadores, y se examinará en la tercera sección. Al final, se describen las cintas magnéticas.

6.1. DISCOS MAGNÉTICOS

Un disco magnético es un plato circular construido con un material no magnético, llamado sustrato, cubierto por un material magnetizable. Tradicionalmente, el sustrato es aluminio o una aleación de aluminio. Recientemente, se han utilizado sustratos de cristal. Los sustratos de cristal tienen una serie de ventajas, entre las cuales se encuentran:

- Mejora en la uniformidad de la superficie magnética para incrementar la fiabilidad del disco.
- Reducción significativa de los defectos en toda la superficie lo que ayuda a reducir los errores de lectura/escritura.
- Capacidad para soportar grabaciones de gran proximidad (*Fly heights*, que se describirán posteriormente).
- Mejor rigidez para reducir la dinámica del disco.
- Mayor capacidad para resistir golpes y daños.

MECANISMOS DE LECTURA Y ESCRITURA MAGNÉTICA

Los datos se graban y después se recuperan del disco a través de una bobina, llamada **cabeza**; en muchos sistemas, hay dos cabezas, una de lectura y otra de escritura.

Durante una operación de lectura o escritura, la cabeza permanece quieta mientras el plato rota bajo ella.

El mecanismo de escritura se basa en el hecho de que un flujo eléctrico atravesando una bobina crea un campo magnético. Se envían pulsos eléctricos a la cabeza de escritura, y se graban los patrones magnéticos en la superficie bajo ella, con patrones diferentes para corrientes positivas y negativas. La propia cabeza de lectura está hecha de un material fácilmente magnetizable y tiene forma de *donut* rectangular con un agujero a lo largo de un lado y varias vueltas de cable conductor a lo largo del lado opuesto (Figura 6.1). Una corriente eléctrica en el cable induce un campo magnético a lo largo del agujero, que magnetiza una pequeña área del medio grabable. Cambiando la dirección de la corriente, cambia el sentido de magnetización del medio de grabación.

El mecanismo tradicional de lectura se basa en el hecho de que un campo magnético en movimiento respecto a una bobina, induce una corriente eléctrica en la bobina. Cuando la superficie del disco pasa bajo la cabeza, en esta se genera una corriente de la misma polaridad que la que produjo la grabación magnética. La estructura de la cabeza de lectura es, este caso, esencialmente la misma que la de escritura y, por tanto, se puede usar la misma cabeza para ambas operaciones. Estas cabezas únicas se usan en disquetes y discos duros antiguos.

Los discos duros de hoy usan un mecanismo diferente para la lectura, siendo necesaria una cabeza de lectura separada posicionada, por conveniencia, cerca de la cabeza de escritura. La cabeza de lectura consiste en un sensor magnetoresistivo (MR) parcialmente blindado. El MR tiene una

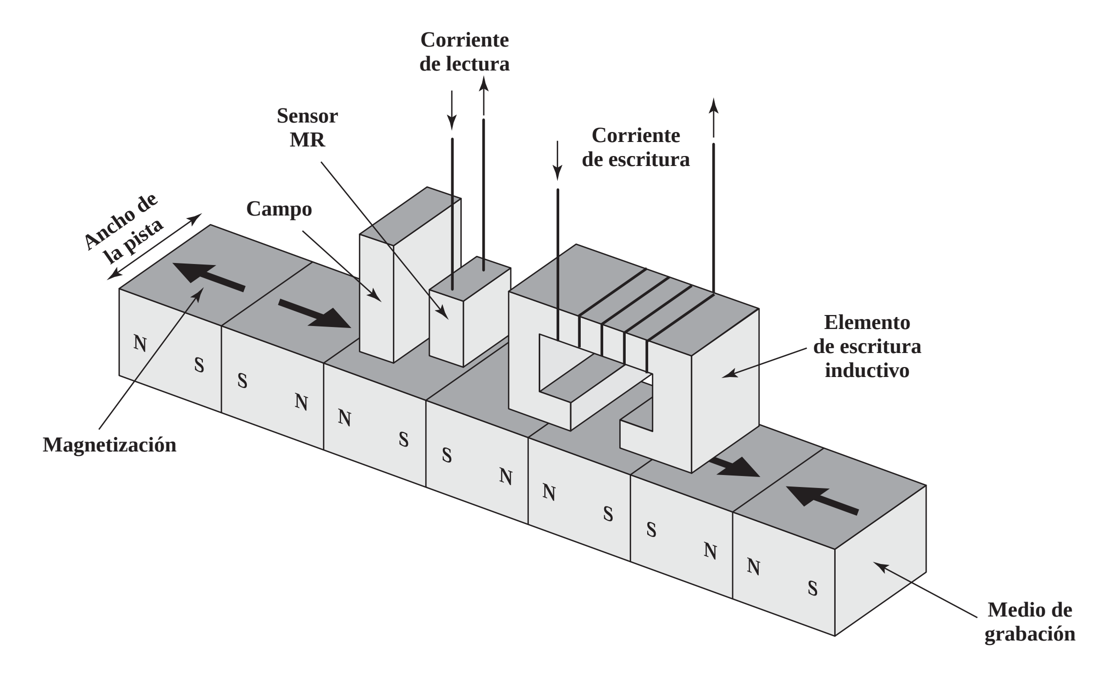


Figura 6.1. Cabeza de (escritura por inducción) / (lectura magnetoresistiva).

resistencia eléctrica que depende de la dirección de la magnetización del medio que se mueve bajo él. Haciendo pasar un corriente a través del sensor MR, los cambios de la resistencia se detectan como señales de tensión. El diseño del MR permite operar a altas frecuencias, lo que lo que equivale a grandes densidades de almacenamiento y de velocidad de funcionamiento.

ORGANIZACIÓN Y FORMATO DE LOS DATOS

La cabeza es un dispositivo relativamente pequeño, capaz de leer o escribir en una zona del plato que rota bajo ella. Esto da lugar a que los datos se organicen en un conjunto de anillos concéntricos en el plato, llamados **pistas**. Cada pista es del mismo ancho que la cabeza. Usualmente hay cientos de pistas por superficie.

En la Figura 6.2 se puede ver la disposición de los datos. Las pistas adyacentes están separadas por **bandas vacías**. Esto previene, o por lo menos minimiza, los errores debidos a desalineamientos de la cabeza o simplemente a interferencias del campo magnético.

Los datos se transfieren al y desde el disco en sectores (Figura 6.2). Normalmente hay cientos de sectores por pista, y estos pueden tener una longitud variable o fija. En la mayoría de los sistemas de hoy se utilizan sectores de longitud fija, siendo 512 bytes el tamaño casi universal de un sector. Para evitar imposiciones de precisión ilógicas del sistema, los sectores adyacentes se separan con intrapistas (intersectores) vacías.

Un bit cercano al centro de un disco girando, pasa por punto fijo (como la cabeza de lectura-escritura) más despacio que un bit más externo. Por tanto, debe haber alguna forma de compensar la

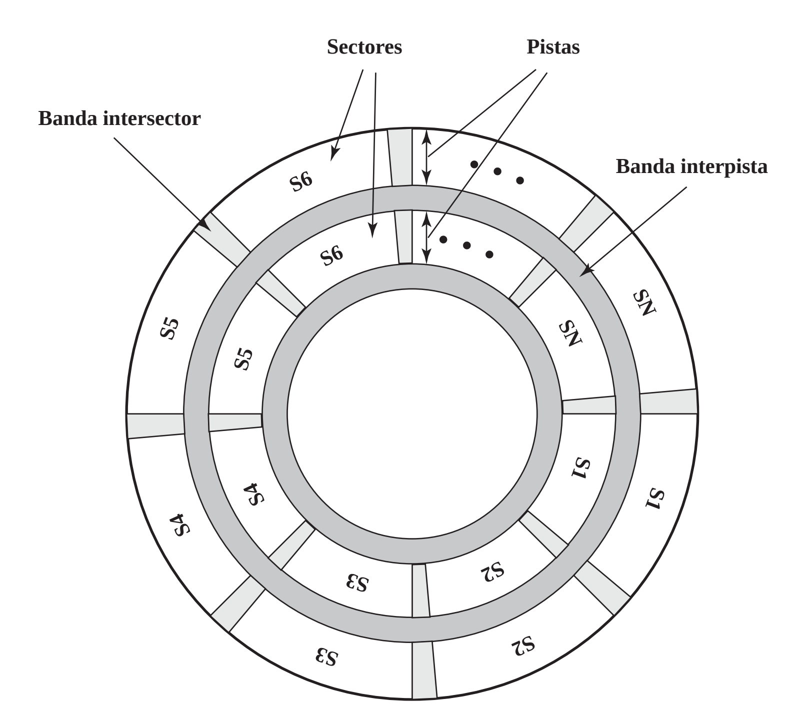
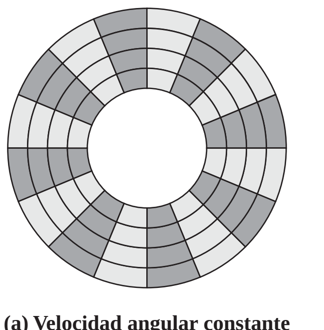


Figura 6.2. Organización de los datos en el disco.



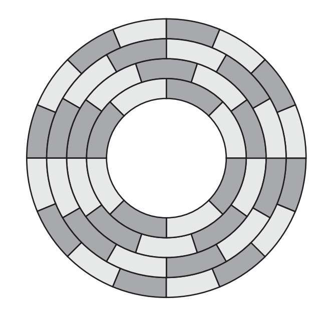


Figura 6.3. Comparación de los métodos de organización de un disco.

variación de la velocidad de forma que la cabeza pueda leer todos los bits a la misma velocidad. Esto se puede hacer incrementando el espacio entre bits de la información grabada en los segmentos del disco. La información se puede escanear a la misma velocidad rotando el disco a una velocidad fija, conocida como **velocidad angular constante** (*constant angular velocity*, CAV). La Figura 6.3a muestra la estructura de un disco que usa CAV. El disco se divide en una serie de sectores en forma de trozo de tarta y en una serie de pistas concéntricas. La ventaja de usar CAV es que los bloques individuales de datos se pueden direccional directamente con la pista y sector. Para mover la cabeza desde su actual posición a una dirección específica, solo hay que mover ligeramente la cabeza a la pista específica y esperar a que el sector se sitúe bajo la cabeza. La desventaja de CAV es que la cantidad de datos que se puede almacenar en las pistas más externas es solo la misma que la de las pistas internas.

Debido a que la **densidad**, en bits por pulgada, aumenta a medida que nos movemos desde la pista más externa a la más interna, la capacidad de almacenamiento de un disco con un sistema CAV sencillo viene limitada por la máxima densidad de grabación que se puede llevar a cabo en la pista más interna. Para aumentar la capacidad, los discos duros modernos utilizan una técnica conocida como **grabación en varias zonas** (*multiple zone recording*), en la que la superficie se divide en varias zonas concéntricas (usualmente 16). Dentro de una zona, el número de bits por pista es constante. Las zonas más lejanas del centro contienen más bits (más sectores) que las zonas próximas al centro. Esto permite capacidades de almacenamiento mayores a expensas de una circuitería de alguna forma más compleja. Como la cabeza del disco se mueve de una zona a otra, la longitud (a lo largo de la pista) de los bits individuales cambia, provocando un cambio en el tiempo de lectura y escritura. La Figura 6.3b sugiere la naturaleza de la grabación en varias zonas; en esta figura, cada zona es una sola pista.

Algún procedimiento es necesario para situar las posiciones del sector en una pista. Claramente, debe haber algún punto de comienzo de la pista y una manera de identificar el principio y el fin de cada sector. Estos requisitos son gestionados mediante datos de control grabados en el disco. Por tanto, el disco se graba con un formato que contiene algunos datos extra usados solo por el controlador del disco y no accesibles al usuario.

En la Figura 6.4 se muestra un ejemplo del formato de grabación de un disco. En este caso, cada pista contiene treinta sectores de longitud fija de 600 bytes cada uno. Cada sector contiene 512 bytes de datos más información de control útil al controlador del disco. El campo ID es un identificador único o dirección usado para localizar un sector particular. El byte SINCRO es un patrón de bits especial que delimita el comienzo del campo. El número de pista identifica una pista en una super-

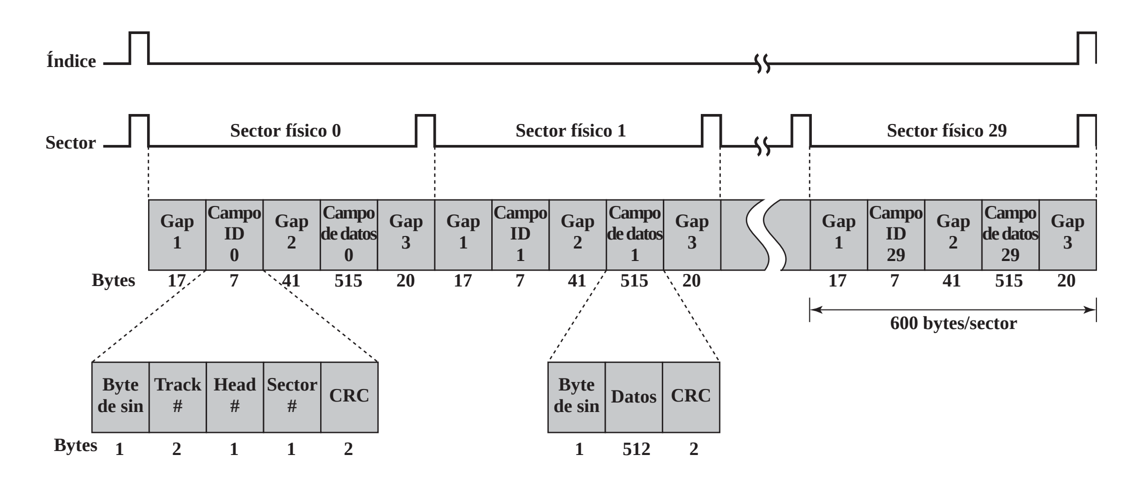


Figura 6.4. Formato de las pistas de un disco Winchester (Seagate ST506).

ficie. El número de cabeza identifica una cabeza, si el disco tiene varias superficies (como acabamos de explicar). El ID y los campos de datos contienen, cada uno, un código de detección de errores.

CARACTERÍSTICAS FÍSICAS

En la Tabla 6.1 se listan las principales características que diferencian los distintos tipos de discos. Primero, las cabezas pueden ser fijas o móviles con respecto a la dirección radial del plato. En un disco de **cabeza fija** hay una cabeza de lectura/escritura por pista. Todas las cabezas se montan en un brazo rígido que se extiende a través de todas las pistas. En un disco de **cabeza móvil**, hay solo una cabeza de lectura/escritura. Como antes, la cabeza se monta en un brazo. Como la cabeza debe poder posicionarse encima de cualquier pista, el brazo debe extenderse o retraerse para este propósito.

El disco mismo, se monta en una unidad de disco, que consta del brazo, un eje que rota el disco, y la electrónica necesaria para la entrada y salida de datos binarios. Un **disco no extraíble** está permanentemente montado en la unidad de disco. Un **disco extraíble**, puede ser quitado y sustituido por otro disco. La ventaja de este último tipo es que es posible una cantidad de datos ilimitada con un

Tabla 6.1. Características físicas de los discos.

Movimiento de la cabeza	**Platos**
Cabeza fija (una por pista)	Un plato
Cabeza móvil (una por superficie)	Varios platos
Portabilidad de los discos	**Mecanismo de la cabeza**
Disco no extraíble	Contacto (disquete)
Disco extraíble	Separación fija
Caras	Separación aerodinámica (Winchester)
Una cara	
Dos caras	

número limitado de unidades de disco. Además, un disco puede ser utilizado en diversos computadores. Los disquetes y los cartuchos ZIP son ejemplos de discos extraíbles.

En la mayoría de los discos, la cubierta magnetizable se aplica a ambas caras del plato, denominándose estos discos de **doble superficie**. Algunos discos, menos caros, son de **una sola superficie**.

Algunas unidades de disco poseen **varios platos** apilados verticalmente y separados por una distancia de alrededor de una pulgada. Disponen de varios brazos (Figura 6.5). Los discos de varios platos utilizan una cabeza que se mueve, con una cabeza de lectura-escritura para cada superficie del plato. El conjunto de todas las pistas que tienen la misma posición relativa en el plato se denomina **cilindro**. Por ejemplo, todas las pistas sombreadas en la Figura 6.6 pertenecen al mismo cilindro.

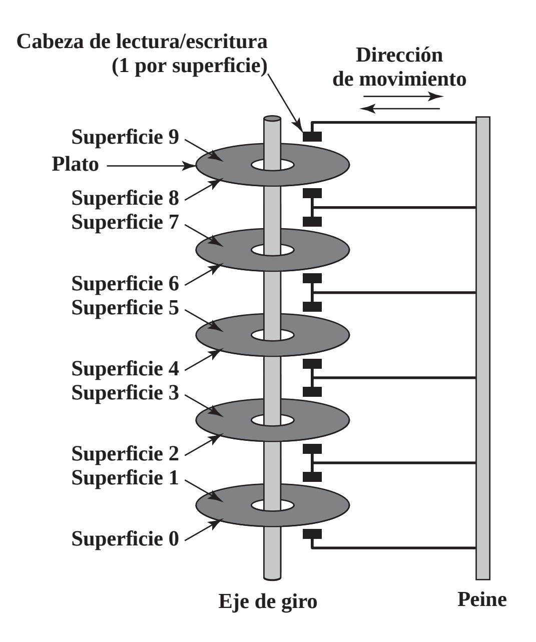


Figura 6.5. Componentes de una unidad de disco.

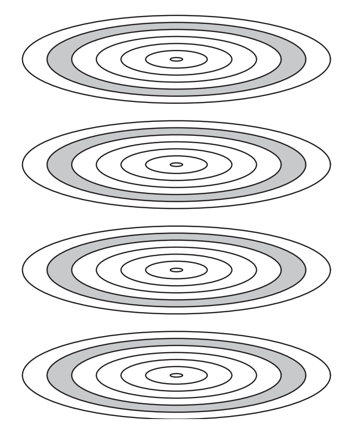


Figura 6.6. Pistas y cilindros.

Finalmente, el mecanismo de la cabeza proporciona una clara clasificación de los discos en tres tipos. Tradicionalmente, la cabeza de lectura/escritura se posiciona a una distancia fija sobre el plato, dejando entre ambos una capa de aire. En el otro extremo está el mecanismo de la cabeza que realmente efectúa un contacto físico con el medio durante la operación de lectura o escritura. Este mecanismo es el que se usa con los **disquetes**, que son pequeños, de plato flexible y es el tipo de disco más barato.

Para entender el tercer tipo de disco, necesitamos comentar la relación entre la densidad de datos y la anchura de la capa de aire. La cabeza debe generar o detectar un campo magnético de intensidad suficiente para escribir y leer correctamente. Cuanto más estrecha es la cabeza, más cercana debe estar a la superficie del plato para funcionar. Esto es deseable, ya que una cabeza más estrecha implica pistas más estrechas y por tanto, mayor densidad de datos. Sin embargo, cuanto más cerca esté la cabeza del disco, mayor será el riesgo de error debido a impurezas o imperfecciones. Los discos Winchester supusieron un avance tecnológico en este sentido.

Las cabezas de los Winchester están montadas en unidades herméticamente cerradas, que están casi libres de contaminación. Fueron diseñados para operar más cerca de la superficie del disco que las cabezas de los discos rígidos anteriores, por tanto permiten una densidad de datos mayor. La cabeza está en el contorno de una hoja de metal aerodinámica que reposa suavemente sobre la superficie del plato cuando el disco no se mueve. La presión del aire generada por el giro del disco es suficiente para hacer subir la hoja encima de la superficie. El sistema sin contacto resultante puede ser diseñado para usar cabezas más estrechas que las de los discos rígidos convencionales, operando más cerca de la superficie de los platos[1].

La Tabla 6.2 muestra los parámetros de los discos de altas prestaciones actuales.

PARÁMETROS PARA MEDIR LAS PRESTACIONES DE UN DISCO

Los detalles de las operaciones de E/S de un disco dependen del tipo de computador, del sistema operativo, de la naturaleza de los canales de E/S y del hardware controlador del disco. En la Figura 6.7 se muestra un diagrama de temporización general de las transferencias de E/S del disco.

Cuando la unidad de disco está funcionando, el disco está rotando a una velocidad constante. Para leer o escribir, la cabeza debe posicionarse en la pista deseada y al principio del sector deseado en la pista. La selección de la pista implica un movimiento de la cabeza, en un sistema de cabeza móvil, o una selección electrónica de una cabeza, en un sistema de cabezas fijas. En un sistema de cabeza móvil, el tiempo que tarda la cabeza en posicionarse en la pista se conoce como **tiempo de búsqueda**. En cualquier caso, una vez seleccionada la pista, el controlador del disco espera hasta que el sector apropiado rote hasta alinearse con la cabeza. El tiempo que tarda el sector en alcanzar la cabeza se llama **retardo rotacional** o latencia rotacional. La suma del tiempo de búsqueda, si lo hay, y el retardo rotacional se denomina **tiempo de acceso**, o tiempo que se tarda en llegar a la posición de lectura o escritura. Una vez posicionada la cabeza, se lleva a cabo la operación de lectura o escritura, desplazándose el sector bajo la cabeza; esta operación conlleva un **tiempo de transferencia de datos**.

[1] Como información de interés histórico, el término Winchester fue usado originalmente por IBM como nombre preliminar para el modelo de disco 3340. El 3340 era un paquete de discos extraíble con las cabezas integradas en el paquete. El término se aplica ahora a cualquier unidad de disco integrada con un diseño de cabezas aerodinámico. El disco Winchester se usa habitualmente en PC y estaciones de trabajo, y se le suele llamar *disco duro*.

Tabla 6.2. Parámetros de las unidades de disco duro.

Características	Seagate Barracuda 180	Seagate Cheetah X15-36LP	Seagate Barracuda 36ES	Toshiba HDD1242	Hitachi Microdrive
Aplicación	Servidor de gama alta capacidad	Servidor de altas prestaciones	Servidor básico	Portátil	Dispositivos de bolsillo
Capacidad	181.6 GB	36.7 GB	18.4 GB	5 GB	4 GB
Tiempo de búsqueda mínimo pista-pista	0.8 ms	0.3 ms	1.0 ms	—	1.0 ms
Tiempo de búsqueda medio	7.4 ms	3.6 ms	9.5 ms	15 ms	12 ms
Velocidad del eje de giro	7200 rpm	15K rpm	7200	4200 rpm	3600 rpm
Retardo rotacional medio	4.17 ms	2 ms	4.17 ms	7.14 ms	8.33 ms
Velocidad máxima de transferencia	160 MB/s	522 a 709 MB/s	25 MB/s	66 MB/s	7.2 MB/s
Bytes por sector	512	512	512	512	512
Sectores por pista	793	485	600	63	—
Pistas por cilindro (número de superficies del plato)	24	8	2	2	2
Cilindros (número de pistas en una cara del plato)	24,247	18,479	29,851	10,350	—

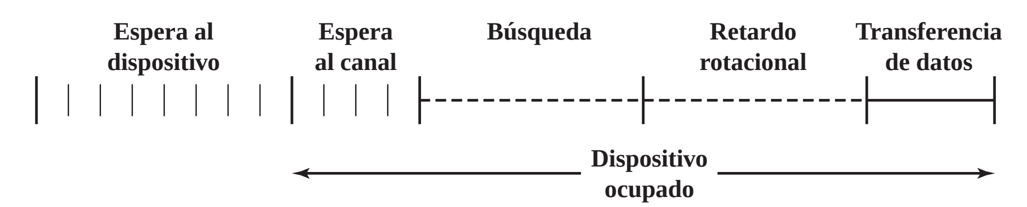


Figura 6.7. Temporizador de la transferencia entre disco y E/S.

Además del tiempo de acceso y de transferencia, hay varios retardos en cola usualmente asociados con operaciones de E/S del disco. Cuando un proceso hace una petición de E/S, primero debe esperar en cola hasta que el dispositivo esté disponible. En ese momento, el dispositivo es asignado al proceso. Si el dispositivo comparte un único canal E/S o un conjunto de canales de E/S con otros discos, entonces puede tener que hacer esperas adicionales para que el canal esté disponible. En este punto se hace la búsqueda para empezar el acceso al disco.

En algunos computadores grandes, se usa una técnica conocida como detección de posición rotacional (RPS, *rotational positional sensing*). Esta funciona de la siguiente forma: cuando se lleva a cabo una orden de búsqueda, el canal es liberado para atender otras operaciones de E/S. Cuando la

búsqueda se ha completado, el dispositivo determina cuándo se rotan los datos bajo la cabeza. Mientras el sector se aproxima a la cabeza, el dispositivo intenta restablecer el camino de comunicación hacia el anfitrión. Si la unidad de control o el canal están ocupados con otra E/S, la conexión puede fallar y el dispositivo debe rotar una vuelta completa antes de que pueda intentar conectarse de nuevo, lo que se denomina una pérdida RPS. Esto supone un retardo extra que se debe añadir a la línea de tiempo de la Figura 6.7.

Tiempo de búsqueda. El tiempo de búsqueda es el tiempo necesario para desplazar el brazo del disco hasta la pista requerida. Este tiempo resulta difícil de precisar. El tiempo de búsqueda está formado por dos componentes clave: el tiempo inicial de comienzo y el tiempo necesario para atravesar las pistas que tienen que cruzarse una vez que el brazo de acceso esté a la velocidad adecuada. El tiempo transversal no es, desgraciadamente, una función lineal del número de pistas, pero incluye un tiempo de espera (tiempo desde que se posiciona la cabeza sobre la pista objetivo hasta que se confirma la identificación de la pista).

Muchas mejoras provienen de componentes más pequeños y ligeros. Hace algunos años, un disco típico tenía 14 pulgadas (36 cm) de diámetro, mientras que hoy el tamaño más normal es de 3,5 pulgadas (8,9 cm), reduciéndose la distancia que tiene que recorrer el brazo. Un tiempo de búsqueda medio típico de un disco actual está entre 100 y 50 ms.

Retardo rotacional. Los discos, que no sean disquetes, rotan a velocidades de 3 600 rpm (para controlar dispositivos como cámaras digitales) en adelante, como 15 000 rpm; esta última velocidad es una revolución cada 4 ms. Por tanto, de media, el retardo rotacional será de unos 2 ms. Las disqueteras normalmente rotan entren 300 y 600 rpm. Por tanto, el retardo medio estará entre los 100 y 50 ms.

Tiempo de transferencia. El tiempo de transferencia hacia o desde el disco depende de la velocidad de rotación del disco de la siguiente forma:

$$T = \frac{b}{rN}$$

donde:

T = tiempo de transferencia

b = número de bytes a transferir

N = número de bytes de una pista

r = velocidad de rotación en revoluciones por segundo

Por tanto, el tiempo de acceso medio total se puede expresar como

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

donde T_s es el tiempo de búsqueda medio. Nótese que en una unidad con zonas, el número de pistas es variable, complicándose el cálculo.

Una comparación de tiempos. Con los parámetros definidos anteriormente, veamos dos operaciones de E/S diferentes que ilustrarán el peligro de fiarse de los valores medios. Considérese un disco con un tiempo de búsqueda medio especificado de 4 ms, una velocidad de rotación de 15 000 rpm, y sectores de 512 bytes con 500 sectores pos pista. Supóngase que queremos leer un fichero

que consta de 2 500 sectores con un total de 1,28 Mb. Queremos estimar el tiempo total de transferencia.

Primero, supongamos que el fichero está almacenado de la forma más compacta posible en el disco. Es decir, el fichero ocupa todos los sectores de 5 pistas adyacentes (5 pistas × 500 sectores/pista = 2 500 sectores). Esto se conoce como *organización secuencial*. Ahora, el tiempo para leer la primera pista es el siguiente:

Búsqueda media	4 ms
Retardo rotacional	2 ms
Leer 500 sectores	4 ms
	10 ms

Supongamos que el resto de las pistas se puede leer ahora sin prácticamente tiempo de búsqueda. Es decir, la operación de E/S puede mantenerse con un flujo continuo desde el disco. Entonces, al menos, se necesita considerar un retardo rotacional para cada pista leída. Entonces, cada pista siguiente se lee en 2 + 4 = 6 ms. Para leer el fichero entero:

$$\text{Tiempo total} = 10 + (4 \times 6) = 34 \text{ ms} = 0{,}034 \text{ segundos}$$

Ahora calculemos el tiempo requerido para leer los mismos datos utilizando acceso aleatorio en vez de secuencial; es decir, los accesos a los sectores se distribuyen aleatoriamente sobre el disco. Para cada sector tenemos:

Búsqueda media	4 ms
Retardo rotacional	2 ms
Leer 1 sector	0,008 ms
	6,008 ms

$$\text{Tiempo total} = 2.500 \times 6{,}008 = 15.020 \text{ ms} = 15{,}02 \text{ segundos}$$

Está claro que el orden en que se lean los sectores desde el disco tiene una repercusión enorme en las prestaciones de E/S. En el caso de acceso a ficheros en los que se lean o escriban varios sectores, se tiene un cierto control sobre la forma en la que los sectores o datos se organizan, y debemos decir algo sobre este tema en el siguiente capítulo. Sin embargo, aún en el caso de un acceso a un fichero, en un entorno de multiprogramación, habrá peticiones de E/S compitiendo por el mismo disco. Entonces, merece la pena examinar maneras en las que las prestaciones de E/S del disco mejoren respecto a las llevadas a cabo con accesos al disco puramente aleatorios. Esto conduce a considerar algoritmos de planificación del disco, que son jurisdicción de los sistemas operativos y están fuera del alcance de este libro (*ver* [STAL05] para más detalles).

6.2. RAID

Como se dijo anteriormente, el ritmo de mejora de prestaciones en memoria secundaria ha sido considerablemente menor que en procesadores y en memoria principal. Esta desigualdad ha hecho, quizá, del sistema de memoria de disco el principal foco de optimización en las prestaciones de los computadores.

Como en otras áreas de rendimiento de los computadores, los diseñadores de memorias de disco reconocen que si uno de los componentes solo se puede llevar a un determinado límite, se puede

conseguir una ganancia en prestaciones adicional usando varios de esos componentes en paralelo. En el caso de la memoria de disco, esto conduce al desarrollo de conjuntos de discos que operen independientemente y en paralelo. Con varios discos, las peticiones separadas de E/S se pueden gestionar en paralelo, siempre que los datos requeridos residan en discos separados. Además, se puede ejecutar en paralelo una única petición de E/S si el bloque de datos al que se va a acceder está distribuido a lo largo de varios discos.

Con el uso de varios discos, hay una amplia variedad de formas en las que se pueden organizar los datos, y en las que se puede añadir redundancia para mejorar la seguridad. Esto podría dificultar el desarrollo de esquemas de bases de datos que se pueden usar en numerosas plataformas y sistemas operativos. Afortunadamente, la industria está de acuerdo con los esquemas estandarizados para el diseño de bases de datos para discos múltiples, conocidos como RAID (*Redundant Array of Independent Disks*, conjunto redundante de discos independientes). El esquema RAID consta de seis niveles[2] independientes, desde cero hasta cinco. Estos niveles no implican una relación jerárquica, sino que designan métodos diferentes que poseen tres características comunes:

1. RAID es un conjunto de unidades físicas de disco vistas por el sistema operativo como una única unidad lógica.
2. Los datos se distribuyen a través de las unidades físicas del conjunto de unidades.
3. La capacidad de los discos redundantes se usa para almacenar información de paridad que garantice la recuperación de los datos en caso de fallo de disco.

Los detalles de las características segunda y tercera cambian según los distintos niveles RAID. RAID 0 no soporta la tercera característica.

El término *RAID* fue originalmente ideado en un artículo de un grupo de investigación de la Universidad de California en Berkley [PATT88][3]. El artículo perfilaba varias configuraciones y aplicaciones RAID e introducía las definiciones de los niveles RAID que todavía se usan. La estrategia RAID reemplaza una unidad de disco de gran capacidad por unidades múltiples de menor capacidad y distribuye los datos de forma que se puedan habilitar accesos simultáneos a los datos de varias unidades mejorando, por tanto, las prestaciones de E/S y permitiendo más fácilmente aumentos en la capacidad.

La única contribución de la propuesta RAID es, efectivamente, hacer hincapié en la necesidad de redundancia. El uso de varios dispositivos, además de permitir que varias cabezas y actuadores operen simultáneamente, consiguiendo mayores velocidades de E/S y de transferencia, incrementa la probabilidad de fallo. Para compensar esta disminución de seguridad, RAID utiliza la información de paridad almacenada que permite la recuperación de datos perdidos debido a un fallo de disco.

A continuación examinaremos cada nivel de RAID. La Tabla 6.3, a partir de [MASS97], proporciona una amplia guía sobre los siete niveles. De ellos, los niveles 2 y 4 no se ofrecen

---

[2] Algunos investigadores y compañías han definido niveles adicionales, pero los seis niveles descritos en esta sección son los convenidos universalmente.

[3] En este artículo, el acrónimo RAID significaba conjunto redundante de discos baratos (*Redundant Array of Inexpensive Disks*). El término barato se usó para contrastar los discos pequeños de los conjuntos RAID, relativamente baratos, frente a la alternativa de discos únicos, grandes y caros (SLED, *Single Large Expensive Disk*). Hoy, el término SLED está obsoleto, y se usan tecnologías similares tanto para configuraciones RAID como no/RAID. De acuerdo con esto, la industria ha adoptado el término *independiente*, para enfatizar que el conjunto RAID proporciona prestaciones adecuadas y mejoras de seguridad.

Tabla 6.3. Niveles RAID.

Categoría	Nivel	Descripción	Discos necesarios	Disponibilidad de los datos	Capacidad de transferencia de datos de E/S alta	Velocidad de petición de E/S baja
Estructuras en tiras	0	No redundante	N	Menor que con un solo disco	Muy alta	Muy alta tanto para lectura como para escritura
Estructura en espejo	1	Xxx	$2N$, $3N$, etc.	Xxxx	xxxx	xxx
Acceso paralelo	2	xxxxx	$N + m$	xxx	xxxx	xxxx
	3	xxxx	$N + 1$	xxxxx	xxxxx	xxxx
Acceso independiente	4	xxxxx	$N + 1$	xxx	xxxx	xxxx
	5	xxxxx	$N + 1$	xxx	xxxx	xxxx
	6	xxxx	$N + 2$	xxxxx	xxxx	xxxx

comercialmente y no es probable que consigan aceptación industrial. En la tabla, las prestaciones de E/S se expresan tanto en términos de la capacidad de transferencia de datos, o capacidad para mover datos, como de la velocidad de petición de E/S, o capacidad de atender las peticiones de E/S, ya que estos niveles RAID operan inherentemente de forma distinta según sean estas dos métricas. El punto fuerte de cada nivel RAID se ha destacado sombreándolo. Las Figuras 6.8 a 6.9 muestran el uso de los siete esquemas RAID, que soportan una capacidad de datos para cuatro discos sin redundancia. En las figuras se destaca la organización de los datos del usuario y de los datos redundantes, y se indican los requisitos de almacenaje relativo de los distintos niveles. Nos referiremos a estas figuras a los largo de la siguiente explicación.

NIVEL 0 DE RAID

El nivel 0 de RAID no es un verdadero miembro de la familia RAID, porque no incluye redundancia para mejorar las prestaciones. Sin embargo, hay algunas aplicaciones, como algunas ejecuciones en supercomputadores, en los que las prestaciones y la capacidad son la preocupación primaria y un costo bajo es más importante que mejorar la seguridad.

Para el RAID 0, los datos del usuario y del sistema están distribuidos a lo largo de todos los discos del conjunto. Esto tiene una notable ventaja frente al uso de un único y gran disco: si hay pendientes dos peticiones diferentes de E/S, para dos bloques de datos diferentes, entonces es muy

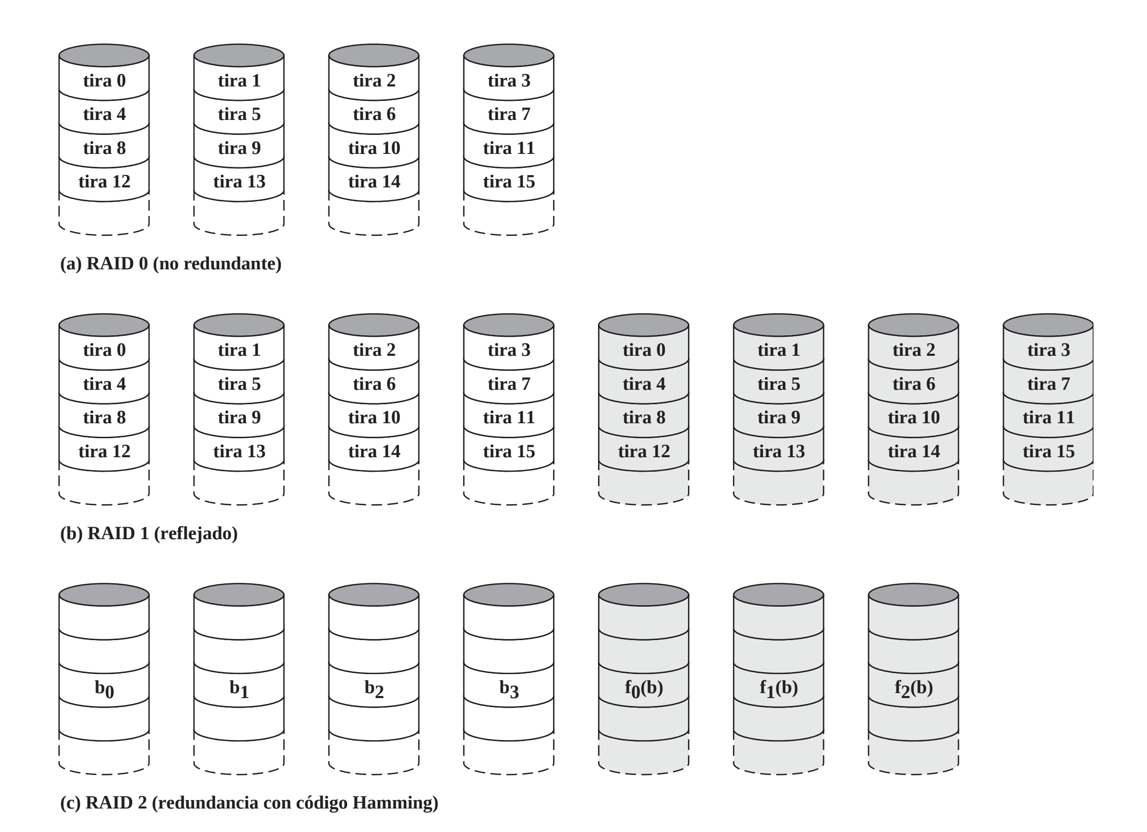


Figura 6.8. Niveles RAID 0 a 3.

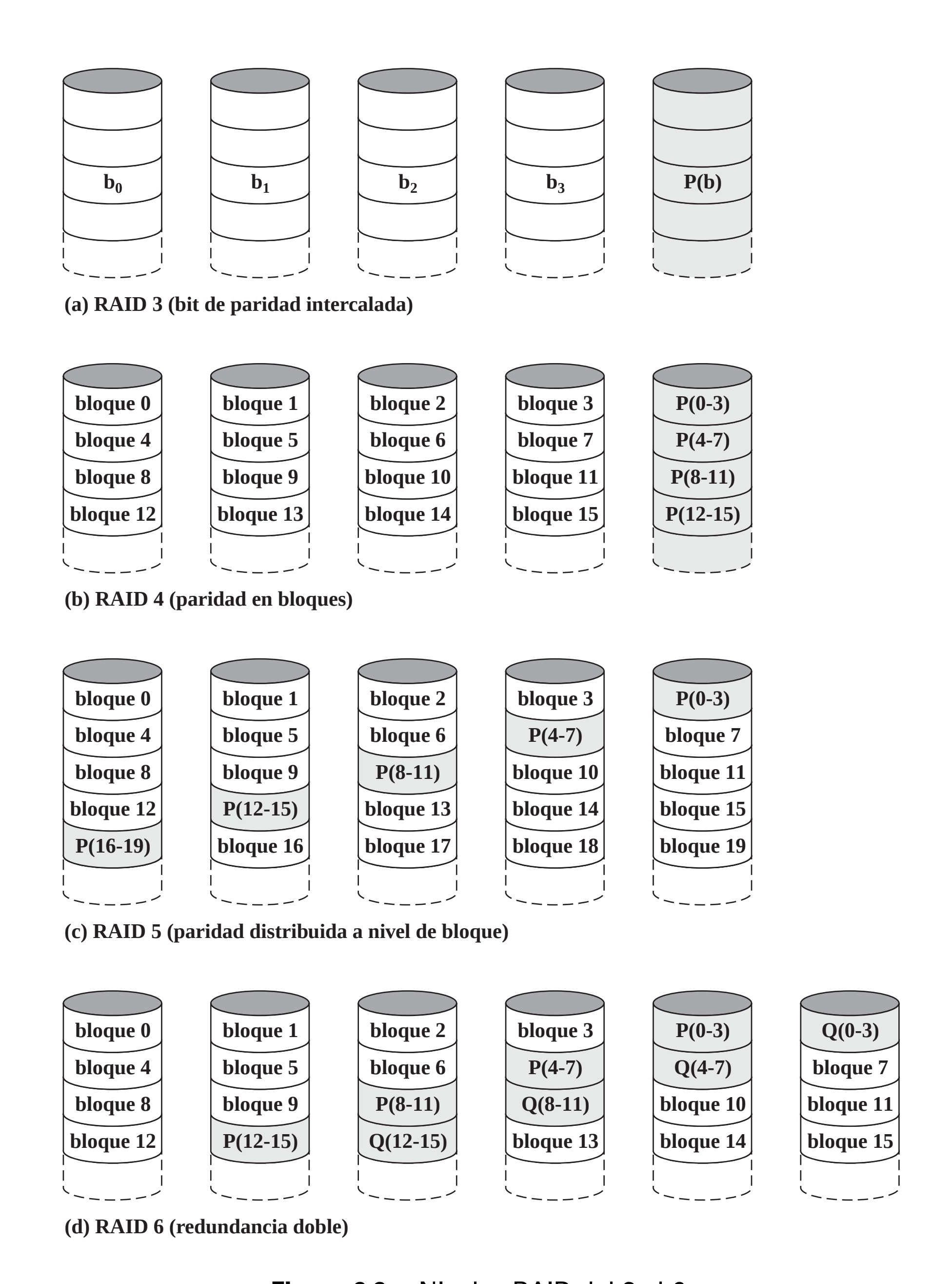


Figura 6.9. Niveles RAID del 3 al 6.

probable que los bloques pedidos estén en diferentes discos. Entonces, las dos peticiones se pueden emitir en paralelo, reduciendo el tiempo de cola de E/S.

Pero RAID 0, como todos los niveles RAID, va más lejos que una sencilla distribución de datos a través del conjunto de discos: los datos son *organizados en forma de tiras de datos* a través de los discos disponibles. Esto se entiende mejor considerando la Figura 6.10. Todos los datos del usuario y del sistema se ven como almacenados en un disco lógico. El disco se divide en tiras; estas tiras pueden ser bloques físicos, sectores o alguna otra unidad. Las tiras se proyectan cíclicamente, en

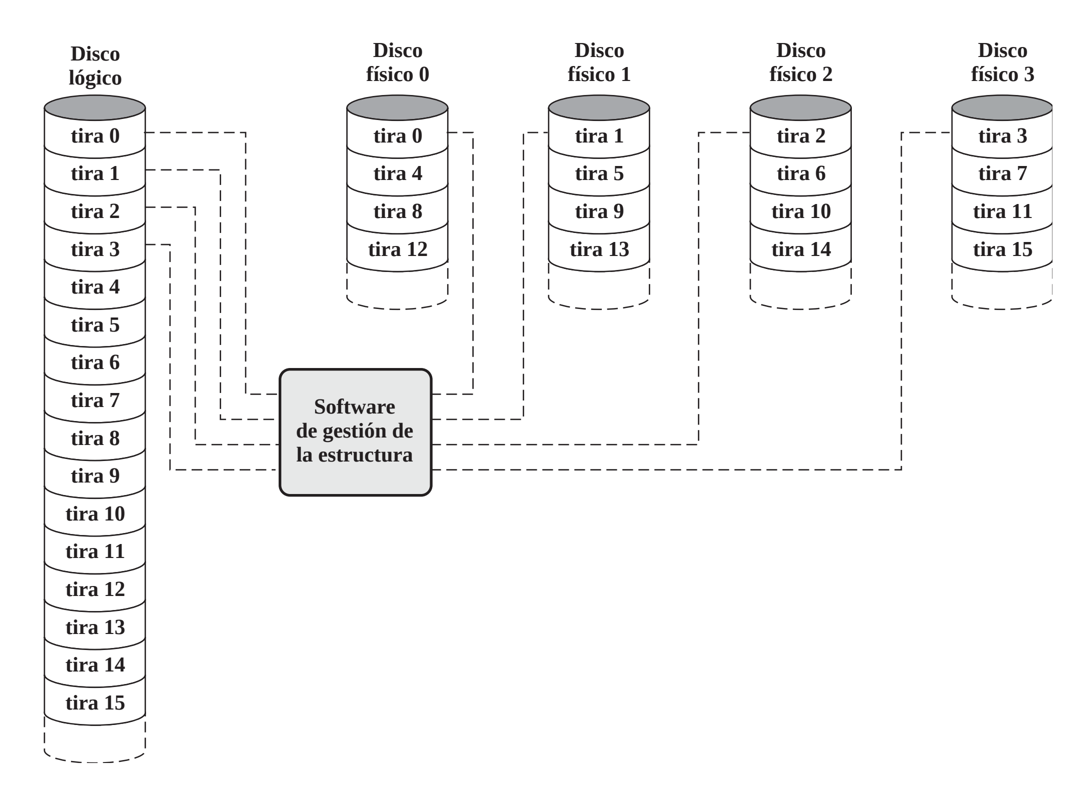


Figura 6.10. Mapa de datos para un conjunto RAID de nivel 0.

miembros consecutivos del conjunto. Un conjunto de tiras lógicamente consecutivas, que se proyectan exactamente sobre una misma tira en cada miembro del conjunto, se denomina franja. En un conjunto de *n* discos, las primeras *n* tiras lógicas (una franja) se almacenan físicamente en la primera tira de cada uno de los *n* discos, las segundas *n* tiras lógicas, se distribuyen en la segunda tira de cada disco, etc. La ventaja de esta disposición es que si una única petición de E/S implica a varias tiras lógicas contiguas, entonces las *n* tiras de esta petición se pueden gestionar en paralelo, reduciendo considerablemente el tiempo de transferencia de E/S.

En la Figura 6.10 se indica como el software de gestión de un conjunto proyecta el espacio del disco físico sobre el disco lógico. Este software se puede ejecutar tanto en el subsistema de disco como en un computador anfitrión.

RAID 0 para alta capacidad de transferencia de datos. Las prestaciones de cualquiera de los niveles RAID dependen críticamente de los patrones de petición del sistema anfitrión y de la distribución de los datos. Estas emisiones pueden ser más claramente direccionadas en RAID 0, donde el impacto de la redundancia no interfiere con el análisis. Primero, consideremos el uso de RAID 0 para lograr una velocidad de transferencia de datos alta. Se deben cumplir dos requisitos para que las aplicaciones tengan una velocidad de transferencia alta. Primero, debe existir una capacidad de transferencia alta en todo el camino entre la memoria del anfitrión y las unidades de disco individuales. Esto incluye controladores de buses internos, buses de E/S del anfitrión, adaptadores de E/S, y buses de memoria del anfitrión.

El segundo requisito es que la aplicación debe hacer peticiones de E/S que se distribuyan eficientemente sobre el conjunto de discos. Esta condición se satisface si la petición típica es de una gran cantidad de datos lógicamente contiguos, comparados con el tamaño de una cinta. En este caso, una única petición de E/S implica la transferencia paralela de datos desde varios discos, aumentando la velocidad efectiva de transferencia, en comparación con la de un único disco.

RAID 0 para altas frecuencias de petición de E/S. En los entornos orientados a transacciones, el usuario se suele preocupar más del tiempo de respuesta que de la velocidad de transferencia. Para una petición individual de E/S de una pequeña cantidad de datos, el tiempo de E/S está dominado por el movimiento de las cabezas del disco (tiempo de búsqueda) y el movimiento del disco (latencia rotacional).

En un entorno de transacción, puede haber cientos de peticiones de E/S por segundo. Un conjunto de discos puede proporcionar velocidades altas de ejecución de E/S, balanceando la carga de E/S a través de los distintos discos. El balanceo de la carga efectiva, se consigue solamente si hay varias peticiones de E/S pendientes. Esto, por turnos, implica que hay varias aplicaciones independientes o una única aplicación orientada a transacción que es capaz de generar varias peticiones de E/S asíncronas. Las prestaciones también se verán influidas por el tamaño de la franja. Si la franja es relativamente grande, de forma que una única petición de E/S solo implique una único acceso a disco, entonces las peticiones de E/S que están esperando pueden ser tratadas en paralelo, reduciendo el tiempo en cola para cada petición.

NIVEL 1 DE RAID

RAID 1 se diferencia de los niveles 2 al 6 en cómo se consigue la redundancia. En estos otros esquemas RAID, se usan algunas formas de cálculo de paridad para introducir redundancia; en RAID 1, la redundancia se logra con el sencillo recurso de duplicar todos los datos. Según muestra la Figura 6.8b, se hace una distribución de datos, como en el RAID 0. Pero en este caso, cada franja lógica se proyecta en dos discos físicos separados, de forma que cada disco del conjunto tiene un disco espejo que contiene los mismos datos. RAID 1 también se puede implementar sin franja de datos, pero es menos común.

En la organización RAID 1 hay una serie de aspectos positivos:

1. Una petición de lectura puede ser servida por cualquiera de los discos que contienen los datos pedidos, cualquiera de ellos implica un tiempo de búsqueda mínimo más la latencia rotacional.
2. Una petición de escritura requiere que las dos tiras correspondientes se actualicen, y esto se puede hacer en paralelo. Entonces, el resultado de la escritura viene determinado por la menos rápida de las dos escrituras (es decir, la que conlleva el mayor tiempo de búsqueda más la latencia rotacional). Sin embargo, en RAID 1 no hay «penalización en la escritura». Los niveles RAID del 2 al 6 implican el uso de bits de paridad. Por tanto, cuando se actualiza una única tira, el software de gestión del conjunto debe calcular y actualizar primero los bits de paridad así como actualizar la tira en cuestión.
3. La recuperación tras un fallo es sencilla. Cuando una unidad falla, se puede acceder a los datos desde la segunda unidad.

La principal desventaja es el coste; requiere el doble del espacio de disco del disco lógico que puede soportar. Debido a esto, una configuración RAID 1 posiblemente está limitada a unidades que almacenan el software del sistema y los datos, y otros ficheros altamente críticos. En estos casos, RAID proporciona una copia de seguridad en tiempo real de todos los datos, de forma que en caso de fallo de disco, todos los datos críticos están inmediatamente disponibles.

En un entorno orientado a transacciones, RAID 1 puede conseguir altas velocidades de petición de E/S si la mayor parte de las peticiones son lecturas. En esta situación, las prestaciones de RAID 1 son próximas al doble de las de RAID 0. Sin embargo, si una parte importante de las peticiones de E/S son peticiones de escritura, entonces la ganancia en prestaciones sobre RAID 0 puede no ser significativa. RAID 1 puede también proporcionar una mejora en las prestaciones de RAID 0 en aplicaciones de transferencia intensiva de datos con un alto porcentaje de lecturas. Se produce una mejora si la aplicación puede dividir cada petición de lectura de forma que ambos miembros del disco participen.

NIVEL 2 DE RAID

Los niveles 2 y 3 de RAID usan una técnica de acceso paralelo. En un conjunto de acceso paralelo, todos los discos miembro participan en la ejecución de cada petición de E/S. Típicamente, el giro de cada unidad individual está sincronizado de forma que cada cabeza de disco está en la misma posición en cada disco en un instante dado.

Como en los otros esquemas RAID, se usa la descomposición de datos en tiras. En el caso de RAID 2 y 3, las tiras son muy pequeñas, a menudo tan pequeñas como un único byte o palabra. Con RAID 2, el código de corrección de errores se calcula a partir de los bits de cada disco, y los bits del código se almacenan en las correspondientes posiciones de bit en varios discos de paridad. Normalmente, se usa el código Hamming, que permite corregir errores en un bit y detectar errores en dos bits.

Aunque RAID 2 requiere menos discos que RAID 1, es todavía bastante caro. El número de discos redundantes es proporcional al logaritmo del número de discos de datos. En una sola lectura, se accede a todos los discos simultáneamente. El controlador del conjunto proporciona los datos pedidos y el código de corrección de errores asociado. Si hay un error en un solo bit, el controlador lo puede reconocer y corregir instantáneamente, con lo que el tiempo de acceso a lectura no se ralentiza. En una escritura sencilla, la operación de escritura debe acceder a todos los discos de datos y de paridad.

RAID 2 debería ser solamente una elección efectiva en un entorno en el que haya muchos errores de disco. Si hay una alta seguridad en los discos individuales y en las unidades de disco, RAID 2 es excesivo y no se implementa.

NIVEL 3 DE RAID

RAID 3 se organiza de manera similar a RAID 2. La diferencia es que RAID 3 requiere solo un disco redundante, sin importar lo grande que sea el conjunto de discos. RAID 3 utiliza un acceso paralelo, con datos distribuidos en pequeñas tiras. En vez de un código de corrección de errores, se calcula un sencillo bit de paridad para el conjunto de bits individuales en la misma posición en todos los discos de datos.

Redundancia. En el caso de un fallo en una unidad, se accede a la unidad de paridad y se reconstruyen los datos desde el resto de los dispositivos. Una vez que se sustituye la unidad que ha fallado, los datos que faltan se restauran en la nueva unidad y se reanuda la operación.

La reconstrucción de los datos es bastante sencilla. Consideremos un conjunto de cinco discos de los que de X0 a X3 contienen datos y X4 es el disco de paridad. La paridad para el i-ésimo bit se calcula de la siguiente forma:

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

donde $\oplus$ es la función exclusive-OR.

Supongamos que la unidad X1 ha fallado. Si sumamos $X4(i) \oplus X1(i)$ a ambos miembros de la ecuación, tenemos que:

$$X1(i) = X4(i) \oplus X3(i) \oplus X2(i) \oplus X0(i)$$

Por lo tanto, se puede regenerar el contenido de cualquier tira de datos en X1 a partir del contenido de las correspondientes tiras del resto de los discos del conjunto. Este principio es válido para los niveles 3 a 6 de RAID.

Caso de que un disco falle, todos los datos estarán todavía disponibles en lo que se denomina modo reducido. En este modo, para lecturas, los datos que faltan se recuperan «al vuelo» con la operación *exclusive-or.* Cuando se escriben datos en un conjunto RAID 3 reducido, se debe mantener la consistencia de la paridad para regeneraciones posteriores. Volviendo al funcionamiento global, se requiere que el disco que ha fallado se reemplace y se regenere todo su contenido en el nuevo disco.

Prestaciones. Puesto que los datos se dividen en tiras muy pequeñas, RAID 3 puede conseguir velocidades de transferencia de datos muy altas. Cualquier petición de E/S implicará una transferencia de datos paralela desde todos los discos de datos. Para grandes transferencias, la mejora de prestaciones es especialmente notable. Por otra parte, solo se puede ejecutar a la vez una petición de E/S. Por tanto, en un entorno orientado a transacciones, el rendimiento sufre.

NIVEL 4 DE RAID

Los niveles 4 al 6 de RAID usan una técnica de acceso independiente. En un conjunto de acceso independiente, cada disco opera independientemente, de forma que peticiones de E/S separadas se atienden en paralelo. Debido a esto, son más adecuados los conjuntos de acceso independiente para aplicaciones que requieren velocidades de petición de E/S altas, y son menos adecuados para aplicaciones que requieren velocidades altas de transferencia de datos.

Como en otros esquemas RAID, se usan tiras de datos. En el caso de RAID 4 a 6, las tiras son relativamente grandes. Con RAID 4, se calcula una tira de paridad bit a bit a partir de las correspondientes tiras de cada disco de datos, y los bits de paridad se almacenan en la correspondiente tira del disco de paridad.

RAID 4 lleva consigo una penalización en la escritura cuando se realiza una petición de escritura de E/S pequeña. Cada vez que se realiza una escritura, el software de gestión del conjunto debe actualizar no solo los datos del usuario, sino también los bits de paridad correspondientes. Consideremos un conjunto de cinco unidades en las que de X0 a X3 contienen datos y X4 es el disco de paridad. Supongamos que se realiza una escritura que implica solo una tira del disco X1.

Inicialmente, para cada bit *i*, tenemos la siguiente relación:

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

Después de la actualización, indicamos con prima los bits que han sido alterados:

$$\begin{aligned} X4'(i) &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \\ &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \oplus X1(i) \oplus X1(i) \\ &= X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i) \oplus X1(i) \oplus X1'(i) \\ &= X4(i) \oplus X1(i) \oplus X1'(i) \end{aligned}$$

El conjunto anterior de ecuaciones se ha obtenido de la siguiente forma. La primera línea muestra que un cambio en X1 también afecta a la paridad del disco X4. En la segundo línea, se han añadido los términos [$\oplus X1(i) \oplus X1(i)$]. Como la XOR de cualquier número consigo mismo es 0, no afecta a la ecuación. Sin embargo, esto se usa por conveniencia, para crear una tercera línea reordenando. Finalmente, la Ecuación (11.1) se usa para reemplazar los cuatro primeros términos por $X4(i)$.

Para calcular la nueva paridad, el software de gestión del conjunto debe leer la antigua tira del usuario y la antigua tira de paridad. Entonces, se pueden actualizar estas dos tiras con nuevos datos y calcular la nueva paridad. Por tanto, cada escritura de una tira implica dos lecturas y dos escrituras.

En el caso de una escritura de E/S de mayor tamaño que implique tiras en todas las unidades de disco, la paridad se puede obtener fácilmente con un cálculo usando solamente los nuevos bits de datos. Por tanto, la unidad de paridad puede ser actualizada en paralelo con las unidades de datos, y no habrá lecturas o escrituras extra.

En cualquier caso, cada operación de escritura implica al disco de paridad, que por consiguiente se convertirá en un cuello de botella.

NIVEL 5 DE RAID

RAID 5 está organizado de manera similar a RAID 4. La diferencia es que RAID 5 distribuye las tiras de paridad a lo largo de todos los discos. Un distribución típica es un esquema cíclico, como se muestra en la Figura 6.9c. Para un conjunto de *n* discos, la tira de paridad está en diferentes discos para las primeras *n* tiras, y este patrón se repite.

La distribución de las tiras de paridad a lo largo de todas las unidades evita el potencial cuello de botella de E/S encontrado en RAID 4.

NIVEL 6 DE RAID

El nivel 6 de RAID se introdujo en un artículo de los investigadores de Berkeley [KATZ89]. En el esquema del nivel 6 de RAID, se hacen dos cálculos de paridad distintos, que se almacenan en bloques separados en distintos discos. Por tanto, un conjunto RAID 6 cuyos datos requieran *N* discos consta de $N + 2$ discos.

La figura 6.9d ilustra este esquema. P y Q son dos algoritmos de comprobación de datos distintos. Uno de los dos calcula la *exclusive-OR* usada en los niveles de 4 y 5 de RAID. Pero el otro es

un algoritmo de comprobación de datos independiente. Esto hace posible la regeneración de los datos incluso si dos de los discos que contienen los datos de los usuarios fallan.

La ventaja del RAID 6 es que proporciona una disponibilidad de los datos extremadamente alta. Tendrían que fallar tres discos en el intervalo MTTR (tiempo medio de reparación) para no poder disponer de los datos. Por otra parte, RAID 6 incurre en una penalización de escritura ya que cada escritura afecta a dos bloques de paridad.

La Tabla 6.4 es un resumen comparativo de los siete niveles.

Tabla 6.4. Comparación de RAID.

Nivel	Ventajas	Inconvenientes	Aplicaciones
0	Las prestaciones de E/S se mejoran mucho repartiendo la carga de E/S entre varios canales unidades. No hay cálculo de paridad de cabecera. Diseño muy sencillo. Fácil de implementar.	El fallo de una sola unidad afectará a todos los datos de una estructura, perdiéndose.	Producción y edición de video. Edición de imágenes. Aplicaciones de pruebas de imprenta. Cualquier aplicación que requiera ancho de banda grande.
1	Una redundancia del cien por cien de los datos implica que no sea necesaria la reconstrucción en caso de fallo de disco, solo una copia del disco a reemplazar. Bajo ciertas circunstancias RAID 1 puede soportar varios fallos de unidades. El diseño del subsistema de almacenamiento RAID más sencillo.	La mayor sobrecarga de todos los tipos RAID (100%) ineficiente.	Contabilidad. Nóminas. Finanzas. Cualquier aplicación que requiera una disponibilidad muy alta.
2	Son posibles velocidades de transferencia de datos extremadamente altas. Cuanto mayor es la velocidad de transferencia requerida, mejor es la relación entre discos de datos y discos ECC. Diseño del controlador relativamente sencillo en comparación con los de los niveles 3, 4 y 5.	Relación muy alta entre discos ECC y discos de datos con tamaños de palabra pequeños (ineficiente). Coste del nivel de entrada muy alto (requisitos de velocidades de transferencia muy altas para justificarlo.	No existen implementaciones comerciales / no es comercialmente viable.
3	Velocidad de transferencia de datos de lectura muy alta. Velocidad de transferencia de datos de escritura muy alta. Un fallo de disco tiene un impacto insignificante en el rendimiento. Una baja relación entre discos ECC (paridad) y discos de datos implica una alta eficiencia.	Velocidad de transacción igual que la de una única unidad de disco como mucho (si lavelocidad de giro está sincronizada). El diseño del controlador es bastante complejo.	Producción de video y secuencias en vivo. Edición de imagen. Edición de video. Aplicaciones de prueba de imprenta. Cualquier aplicación que requiera un alto rendimiento.

(Continúa)

Tabla 6.4. Comparación de RAID *(continuación).*

Nivel	Ventajas	Inconvenientes	Aplicaciones
4	Velocidad de transacción de datos de lectura muy alta. Una baja relación entre discos ECC (paridad) y discos de datos implica una alta eficiencia.	Diseño del controlador bastante complejo. Peor velocidad de transacción de escritura y velocidad de transferencia total de escritura. Reconstrucción de datos difícil e ineficiente en caso de fallo de disco.	No existe implementación comercial / no es comercialmente viable.
5	La mayor velocidad de transacción de datos. Una baja relación entre discos ECC (paridad) y discos de datos implica una alta eficiencia. Buena velocidad de transferencia en su conjunto.	Diseño del controlador más complejo. Es difícil la reconstrucción en caso de fallo de disco (comparado con el nivel 1 de RAID).	Servidores de ficheros y aplicaciones. Servidores de bases de datos. Servidores de páginas web, correo electrónico y noticias. Servidores Intranet. Nivel RAID más versatil.
6	Proporciona una tolerancia a fallos extremadamente alta y puede soportar varios fallos de unidades simultáneos.	Diseño del controlador más compleja. La sobrecarga del controlador para calcular las direcciones de paridad es extremadamente alta.	Solución perfecta para aplicaciones con objetivos críticos.

6.3. MEMORIA ÓPTICA

En 1983, se introdujo uno de los productos de consumo de más éxito de todos los tiempos: el disco compacto (CD, *Compact Disk*) digital de audio. El CD es un disco no borrable que puede almacenar más de sesenta minutos de información de audio en una cara. El gran éxito comercial del CD posibilitó el desarrollo de la tecnología de discos de memoria óptica de bajo coste, que revolucionó el almacenamiento de datos en un computador. Se han introducido una gran variedad de discos ópticos (Tabla 6.5). Vamos a ver cada uno de ellos brevemente.

DISCOS COMPACTOS

CD-ROM. Tanto el CD de audio como el CD-ROM (*compact disk read-only memory,* memoria de disco compacto de solo-lectura) comparten una tecnología similar. La principal diferencia es que los lectores de CD-ROM son más robustos y tienen dispositivos de corrección de errores para asegurar que los datos se transfieren correctamente del disco al computador. Ambos tipos de disco se hacen también de la misma forma. El disco se forma a partir de una resina, como un policabornato.

Tabla 6.5. Discos ópticos.

CD
Disco compacto. Un disco no borrable que almacena información de audio digitalizada. El sistema estándar usa discos de doce cm y puede grabar más de sesenta minutos de tiempo de ejecución ininterrumpido.

CD-ROM
Disco compacto de memoria de solo-lectura. Un disco no borrable usado como memoria de datos de un computador. El sistema estándar usa discos de doce cm y puede guardar más de 650 MB.

DVD
Disco versátil digital. Una tecnología para producir representación de información de vídeo digitalizada y comprimida, así como grandes cantidades de otros datos digitales. Se usan en formatos de ocho y doce cm de diámetro, con una capacidad con doble cara de hasta 17 GB. El DVD básico es de solo-lectura (DVD-ROM).

DVD-R
DVD grabable. Es similar al DVD-ROM. El usuario puede escribir en el disco solo una vez. Solo se utilizan discos de una cara.

DVD-RW
DVD grabable. Es similar al DVD-ROM. El usuario puede borrar y reescribir el disco varias veces. Solo se utilizan discos de una cara.

La información grabada digitalmente (ya sea música o datos del computador) se graba como una serie de hoyos microscópicos en la superficie reflectante. Esto se hace, primero de todo, con un láser de alta intensidad y enfocado con precisión, para crear el disco patrón. El patrón se usa, sin embargo para hacer una matriz para estampar copias en policarbonato. La superficie con los hoyos se cubre con una superficie altamente reflectante como aluminio u oro. Esta superficie brillante se protege contra el polvo y los arañazos con una última capa de laca transparente. Finalmente, se puede imprentar una etiqueta sobre la laca.

La información del CD o CD-ROM se recupera con un láser de baja potencia situado en un lector o unidad de disco óptico. El láser pasa a través de la capa protectora transparente mientras un motor hace girar el disco sobre el láser (Figura 6.11). La intensidad de la luz reflejada cambia si se encuentra un hoyo. En concreto, si el haz de láser cae sobre un hoyo, que de alguna manera es una superficie rugosa, la luz se dispersa y una luz de baja intensidad llega a la fuente. Las áreas entre hoyos se llaman *valles*. Un valle es una superficie lisa, que refleja con mayor intensidad. El cambio entre hoyos y valles es detectado por un fotosensor y convertido en una señal digital. El sensor barre la superficie a intervalos regulares. El principio o fin de un hoyo representa un 1; cuando no hay cambios en la altura entre intervalos, se graba un 0.

Recordemos que en un disco magnético, la información se graba en pistas concéntricas. Con el sistema de velocidad angular constante (CAV), el número de bits por pista es constante. Se puede conseguir un incremento en la densidad con la grabación de varias zonas, en la que la superficie se divide en una serie de zonas, de forma que las zonas lejanas al centro contienen más bits que las zonas cercanas al mismo. Aunque esta técnica incrementa la capacidad, no está todavía optimizada.

Para conseguir mayor capacidad, los CD y CD-ROM no se organizan en pistas concéntricas. En su lugar, el disco contiene una única pista en espiral, que comienza en el centro y se extiende hacia el

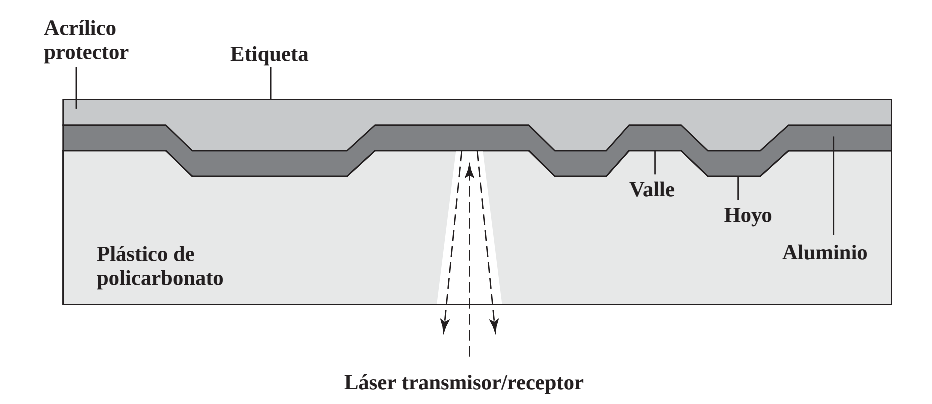


Figura 6.11. Funcionamiento de un CD.

borde del disco. Los sectores cercanos al filo del disco tienen la misma longitud que los cercanos al centro. Por tanto, la información está empaquetada uniformemente a lo largo del disco en segmentos del mismo tamaño y son escaneados a la misma velocidad rotando el disco a velocidad variable. Los hoyos son leídos por un láser a **velocidad lineal constante (CLV)**. El disco rota más despacio en los accesos cercanos al filo externo que en los cercanos al centro. Por tanto, la capacidad de una pista y el retardo rotacional es mayor cercano al centro. La capacidad de un CD-ROM es de unos 650 MB.

Los datos de un CD-ROM se organizan en una secuencia de bloques. En la Figura 6.12 se muestra un formato típico de un bloque. Este consta de los siguientes campos:

- **Sincronización:** el campo de sincronización identifica el principio de un bloque. Consta de un byte de 0s, 10 bytes de 1s, y un byte de 0s.
- **Cabecera:** la cabecera contiene la dirección del bloque y el byte de modo. El modo 0 especifica un campo de datos en blanco; el modo 1 especifica el uso de un código de corrección de errores y 2048 bytes de datos; el modo 2 especifica 2336 bytes de datos del usuario sin código de corrección de errores.
- **Datos:** datos del usuario.
- **Auxiliar:** datos del usuario adicionales, en modo 2. En modo 1, es un código de corrección de errores de 288 bytes.

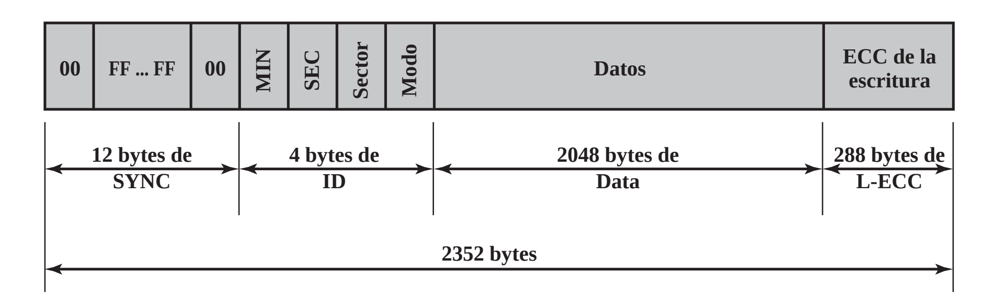


Figura 6.12. Formato de los bloques de un CD-ROM.

Usando CLV, el acceso aleatorio se hace más difícil. Localizar una dirección específica implica mover la cabeza al área general, ajustar la velocidad de rotación y leer la dirección, y hacer pequeños ajustes para encontrar y acceder al sector específico.

Los CD-ROM son apropiados para la distribución de grandes cantidades de datos a un gran número de usuarios. Debido al gasto del proceso inicial de escritura, no es adecuado para aplicaciones individuales. Comparado con los discos magnéticos tradicionales, el CD-ROM tiene dos ventajas:

- El disco óptico junto con la información almacenada en él, se puede replicar en grandes cantidades de forma barata (a diferencia de los discos magnéticos). Las bases de datos en un disco magnético se reproducen copiando uno a uno, usando dos unidades de disco.
- El disco óptico es extraíble, permitiendo usar el mismo disco como memoria de archivo. La mayoría de los discos magnéticos no son extraíbles. La información que contiene tiene que copiarse en una cinta antes de que se pueda usar la unidad de disco/disco para almacenar nueva información.

Las desventajas del CD-ROM son:

- Es de solo lectura y no se puede actualizar.
- El tiempo de acceso es mayor que el de una unidad de disco magnético, tanto como medio segundo.

CD grabable. Para adaptarse a aplicaciones en las que solo se necesitan unas pocas copias de un conjunto de datos, se han desarrollado los CD de una escritura y varias lecturas, conocido como CD grabable (CD-R, CD-*recordable*). Para hacer un CD-R, se prepara un disco de forma que se pueda escribir una vez con un haz láser de intensidad modesta. De esta forma, con algún controlador de disco especial, más caro que para CD-ROM, el cliente puede escribir una vez, además de leer el disco.

El material del CD-R es similar pero no idéntico al de un CD o CD-ROM. En los CD y CD-ROM, la información se graba haciendo pequeños agujeros en la superficie del material, de forma que cambie su reflectividad. En un CD-R, el medio incluye una capa de tinte. El tinte se utiliza para cambiar la reflectividad y se activa con un láser de alta intensidad. El disco resultante se puede leer en una unidad de CD-R o CD-ROM.

El disco óptico CD-R resulta atractivo como almacén de documentos y ficheros. Proporciona una copia permanente para gran cantidad de datos.

CD regrabable. El disco óptico CD-RW se puede escribir y reescribir como un disco magnético. A pesar de las numerosas técnicas que se han probado, la única puramente óptica que ha conseguido ser atractiva se denomina **cambio de fase**. El disco de cambio de fase utiliza un material que presenta dos tipos de reflexión, significantemente diferentes, en dos estados diferentes. Hay un estado amorfo, en el que las moléculas presentan una orientación aleatoria y que refleja mal la luz; y un estado cristalino, que presenta una superficie lisa que refleja bien la luz. Un haz de láser puede cambiar el material de una fase a otra. La principal desventaja del cambio de fase de los discos ópticos es que el material finalmente y de forma permanente pierde sus propiedades. Los materiales actuales se pueden borrar entre 500 000 y un millón de veces.

Los CD-RW tienen la ventaja obvia sobre los CD-ROM y CD-R que se pueden regrabar y por tanto usarse como verdaderos almacenes secundarios. Por tanto, compiten con los discos

magnéticos. Una ventaja clave de los discos ópticos es que las tolerancias de los parámetros de construcción en los discos ópticos es mucho más severa que para los discos magnéticos de gran capacidad. Por tanto, tienen una mayor fiabilidad y vida.

DISCO DIGITAL VERSÁTIL

Con la gran capacidad de almacenamiento del disco digital versátil (DVD, *Digital Video Disk*), la industria de la electrónica ha encontrado por fin un sustituto razonable de las cintas VHS de vídeo analógicas. El DVD sustituirá a las cintas de vídeo usadas en los reproductores de vídeo (VCR) y, lo que es más importante para este texto, sustituirá al CD-ROM en los PC y servidores. El DVD lleva al vídeo a la edad digital. Proporciona películas con una calidad de imagen impresionante, y se puede acceder a ellos aleatoriamente como en los CD de audio, que pueden también pueden leer los DVD. En un disco se puede grabar un gran volumen de datos, en la actualidad siete veces más que en un CD-ROM. Con esta gran capacidad de almacenamiento y alta calidad de los DVD, los juegos para PC serán más reales y el software educativo incorporará más vídeo. Como consecuencia de estos desarrollos habrá un nuevo pico en el tráfico en Internet e intranets corporativas, ya que este material se incorporará a los sitios web.

La mayor capacidad del DVD se debe a tres diferencias respecto al CD (Figura 6.13):

1. Los bits se empaquetan más juntos en un DVD. El espacio entre las vueltas de una espiral en un CD es de 1,6 μm y la distancia mínima entre hoyos a lo largo de la espiral es de 0,834 μm.

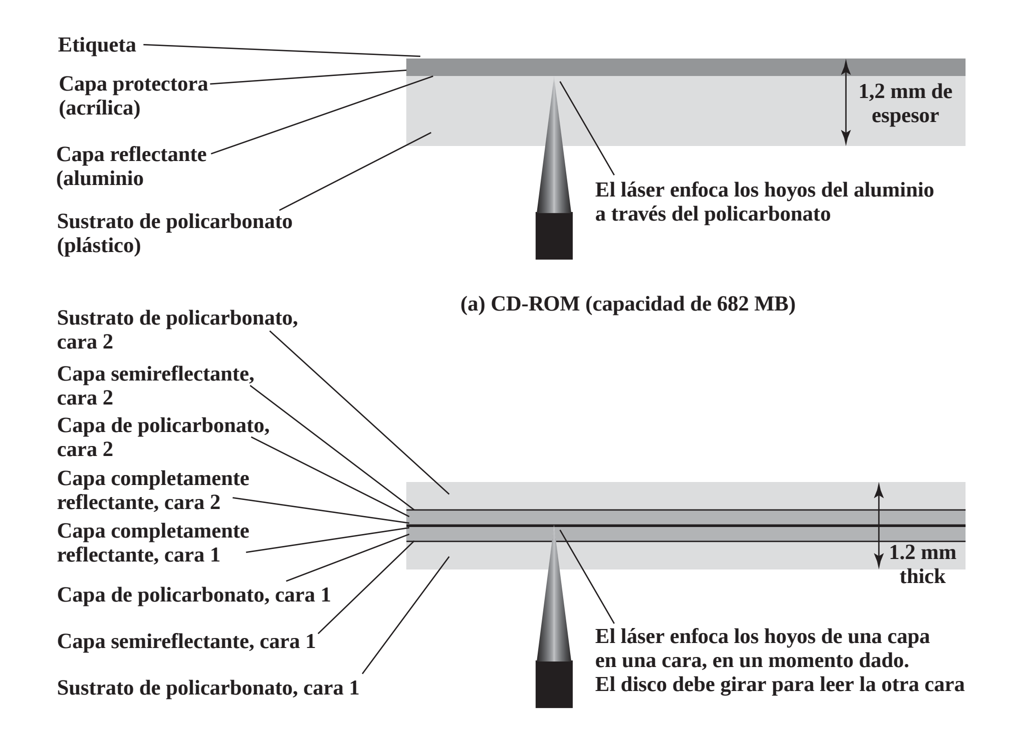


Figura 6.13. CD-ROM y DVD-ROM.

El DVD utiliza un láser con una longitud de onda menor y consigue un espaciado entre vueltas de 0,74 μm y una distancia mínima entre hoyos de 0,4 μm.
El resultado de estas dos mejoras supone un incremento de capacidad en un factor de siete, de alrededor de 4,7 GB.

2. El DVD utiliza una segunda capa de hoyos y valles sobre la primera capa. Un DVD de doble capa tiene una capa semirreflectante sobre la capa reflectante, y, ajustando el enfoque, el láser de la unidad de DVD puede leer cada capa por separado. Esta técnica casi dobla la capacidad del disco, hasta 8,5 GB. La baja reflectividad de la segunda capa limita su capacidad de almacenamiento por lo que no se consigue doblar la capacidad.
3. El DVD-ROM puede tener dos superficies, mientras que en un CD los datos se graban solo en una superficie. Esto da una capacidad total de más de 17 GB.

Como con los CD, los DVD tienen versiones grabables y de solo-lectura (Tabla 6.5).

6.4. CINTA MAGNÉTICA

Los sistemas de cinta usan las mismas técnicas de lectura y grabación que los discos. El medio es una cinta de poliéster flexible (parecido al usado en ropa) cubierta por un material magnetizable. La cubierta puede consistir en partículas de un metal puro en concreto un revestimiento o película de metal plateado vaporizado. La cinta y la unidad de cinta son análogas a las cintas de grabación domésticas. Los anchos de las cintas pueden variar entre 0,38 cm (0,15 pulgadas) y 1,27 cm (0,5 pulgadas). Una cinta ubicada en un carrete abierto tienen que enrollarse en otro carrete ubicado en un segundo cabezal. Hoy día, prácticamente todas las cintas vienen cerradas en cartuchos.

Los datos en la cinta, se estructuran en una serie de pistas paralelas longitudinales. Los primeros sistemas de cintas usaban nueve pistas. Esto hace posible almacenar datos de un byte en un instante dado, con un bit de paridad adicional, en la novena pista. Los nuevos sistemas de cintas usan 18 o 36 pistas, correspondiendo a una palabra o doble palabra digital. La grabación de datos de esta forma se denomina **grabación paralela**. Los sistemas más modernos utilizan en su lugar **grabación serie**, en la que los datos se disponen como una secuencia de bits a lo largo de cada pista, como se hace en los discos magnéticos. Como con el disco, los datos se leen y escriben en bloques contiguos, llamados *registros físicos* de cinta. Los bloques en la cinta están separados por bandas vacías llamadas bandas *inter-registros*. Como en el disco, la cinta se formatea para facilitar la localización de los registros físicos.

La técnica típica utilizada en la grabación de cintas en serie se denomina **grabación en serpentina**. En esta técnica, cuando se graban los datos, el primer conjunto de bits se graba a lo largo de toda la cinta. Cuando se alcanza el fin, las cabezas se posicionan para grabar una nueva pista y la cinta se graba de nuevo a todo lo largo, esta vez en dirección contraria. Este proceso continua, hacia atrás y hacia delante, hasta que la cinta se llena (Figura 6.14a). Para aumentar la velocidad, la cabeza de lectura-escritura es capaz de leer y escribir una serie de pistas adyacentes simultáneamente (usualemente entre dos y ocho pistas). Los datos se graban en serie a lo largo de las pistas individuales, pero los bloques se almacenan en pistas adyacentes, como se sugiere en la Figura 6.14b. La Tabla 6.6 muestra los parámetros de un sistema, conocido como cinta DLT.

Una unidad de cinta es un dispositivo de *acceso secuencial*. Si la cabeza de la cinta se posiciona en el registro 1, entonces para leer el registro *N*, es necesario leer los registros físicos del 1 al *N*-1,

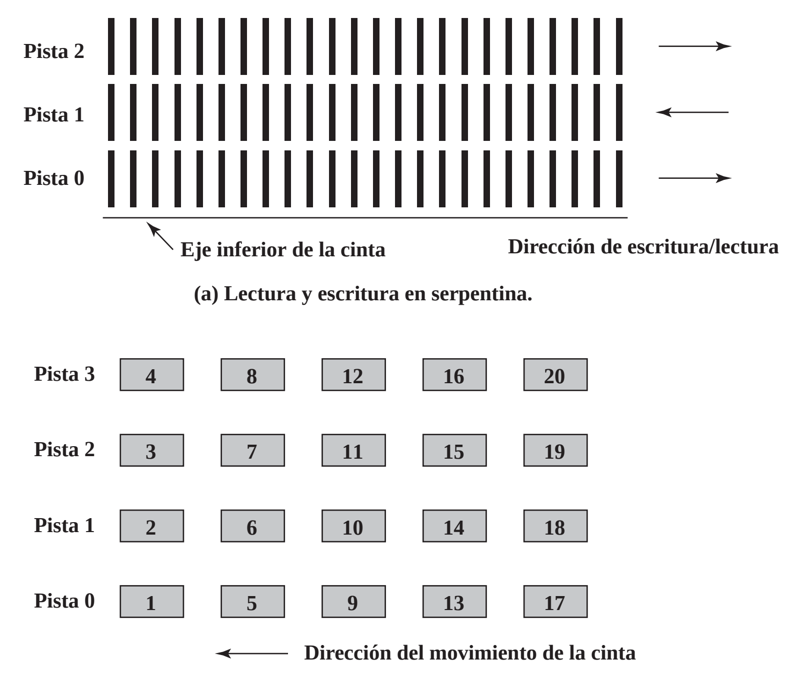


Figura 6.14. Características de una cinta magnética típica.

Tabla 6.6. Unidades de cintas DLT.

	DLT 4000	DLT 8000	SDLT 600
Capacidad (GB)	20	40	300
Velocidad de transferencia (MB/s)	1.5	6.0	36.0
Densidad de bits (Kb/cm)	32.3	38.6	92
Densidad de pistas (p/cm)	101	164	587
Longitud media (m)	549	549	597
Ancho medio (cm)	1.27	1.27	1.27
Número de pistas	128	208	448
Número de pistas de lectura/escritura simultáneas	2	4	8

uno a uno. Si la cabeza está actualmente situada más allá del registro deseado, es necesario rebobinar la cinta una cierta distancia y empezar a leer hacia delante. A diferencia del disco, la cinta está en movimiento solamente durante las operaciones de lectura o escritura.

En contraste con las cintas, a la unidad de disco se le llama dispositivo de *acceso directo*. Una unidad de disco no necesita leer todos los sectores de un disco secuencialmente para llegar al sector deseado.

Solo debe esperar a los sectores dentro de una pista y puede hacer accesos sucesivos a cualquier pista.

Las cintas magnéticas fueron el primer tipo de memorias secundarias. Se usan todavía ampliamente como los miembros de la jerarquía de memoria de menor coste y de menor velocidad.

6.5. LECTURAS Y SITIOS WEB RECOMENDADOS

[MEE96a] ofrece un buen resumen de la tecnología de grabación subyacente de los discos y cintas. [MEE96b] se centra en las técnicas de almacenamiento de datos en discos y cintas.

[COME00] es un artículo corto pero instructivo sobre las tendencias actuales en tecnologías de almacenamiento en discos magnéticos.

Un excelente estudio sobre la tecnología RAID, escrito por los inventores del concepto RAID, está en [CHEN94]. Un buen artículo resumen es [FRIE96]. Una buena comparación de las arquitecturas RAID se encuentra en [CHEN96].

[MARC90] da una excelente visión del campo de las memorias ópticas. Un buen examen de las tecnologías subyacentes de grabación y de lectura es [MANS97].

[ROSC03] proporciona una visión comprensiva de todos los tipos de memorias externas, con una modesta cantidad de detalles técnicos de cada uno. [KHUR01] es otra buena revisión.

ANDE03 Anderson, D.: «You Don't Know Jack About Disks». *ACM Queue,* junio, 2003.

CHEN94 Chen, P.; Lee, E.; Gibson, G.; Katz, R. y Patterson, D.: «RAID: High: High-Performance, Reliable Secondary Storage». *ACM Compuying Surveys,* junio, 1994.

CHEN96 Chen, S. y Towsley, D.: «A Performance Evaluation of RAID Architectures». *IEEE Transactions on Computers,* octubre, 1996.

COME00 Comerfonrd, R.: «Magnetic Storage: The Medium that Wouldn't Die». *IEEE Spectrum,* diciembre, 2000.

FRIE96 Friedman, M.: «RAID Keeps Going and Going and ...». *IEEE Spectrum,* abril, 1996.

KHUR01 Khurshudov, A.: *The Essential Guide to Computer Data Storage. Upper* Saddle River, NJ. Prentice Hall, 2001.

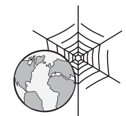
MANS97 Mansuripur, M. y Sincerbox, G.: «Principles and Techniques of Optical Data Storage». *Proceedings of the IEEE,* noviembre, 1997.

MAR90 Marchant, A.: *Optical Recording.* Reading, MA. Addison-Wesley, 1990.

MEE96a Mee, C. y Daniel, E. eds.: *Magnetic Recording Technology.* New York. McGraw-Hill, 1996.

MEE96b Mee, C. y Daniel, E. eds.: *Magnetic Storage Handbook.* New York. McGraw-Hill, 1996.

ROSC03 Rosch, W.: *Winn L. Rosch Hardware Bible.* Indianapolis, IN. Que Publishing, 2003.



SITIOS WEB RECOMENDADOS

- **Asociación de tecnología óptica de almacenamiento:** buena fuente de información sobre tecnologías de almacenamiento óptico y vendedores, más una extensa lista de enlaces importantes.
- **Cintas DLT:** buena colección de información técnica y enlaces a vendedores.

6.6. PALABRAS CLAVE, PREGUNTAS DE REPASO Y PROBLEMAS

PALABRAS CLAVE

banda	disco magnético	pista
cabeza	disco no extraíble	Plato
CD	disquete	RAID
CD-R	DVD	retardo rotacional
CD-ROM	DVD-R	Sector
CD-RW	DVD-ROM	sustrato
cilindro	DVD-RW	tiempo de acceso
cinta magnética	grabación en serpentina	tiempo de búsqueda
datos divididos	Grabación en varias zonas	tiempo de transferencia
disco de cabeza extraíble	hoyo	valle
disco de cabeza fija	magnetorresistivo	velocidad angular constante (CAV)
disco extraíble	memoria óptica	velocidad lineal constante (CLV)

PREGUNTAS DE REPASO

6.1. ¿Cuáles son las ventajas de usar un sustrato de cristal en un disco magnético?
6.2. ¿Cómo se escriben los datos en un disco magnético?
6.3. ¿Cómo se leen los datos en un disco magnético?
6.4. Explicar la diferencia entre un sistema de grabación CAV y de varias zonas.
6.5. Definir los términos *pista, cilindro y sector.*
6.6. ¿Cuál es el tamaño típico de un sector en un disco?
6.7. Definir los términos *tiempo de búsqueda, retardo rotacional, tiempo de acceso* y *tiempo de transferencia.*
6.8. ¿Qué características comunes comparten todos los niveles RAID?
6.9. Definir brevemente los siete niveles RAID.
6.10. Explicar el término *datos divididos.*
6.11. ¿Cómo se consigue redundancia en un sistema RAID?
6.12. En el contexto de RAID, ¿cuál es la diferencia entre acceso paralelo y acceso independiente?
6.13. ¿Cuál es la diferencia entre CAV y CLV?
6.14. ¿En qué se diferencia un CD de un DVD en lo que respecta a la capacidad de este último?
6.15. Explicar la grabación en serpentina.

PROBLEMAS

6.1. Considérese un disco con N pistas numeradas desde 0 hasta $(N - 1)$ y suponer que los sectores requeridos están distribuidos aleatoria y uniformemente a lo largo del disco. Calcular el número medio de pistas atravesadas en una búsqueda.

(a) Primero, calcular la probabilidad de una búsqueda de longitud *j* en la que la cabeza está posicionada en la pista *t*. *Ayuda:* se trata de determinar el número total de combinaciones, reconociendo que todas las pistas son destinos igualmente probables.

(b) Después, calcular la probabilidad de una búsqueda de longitud *K*. *Ayuda:* esto implica sumar todas las posibles combinaciones de movimientos de las *K* pistas.

(c) Calcular el número medio de pistas atravesadas en una búsqueda, usando la siguiente fórmula del valor esperado:

$$E[x] = \sum_{i=0}^{N-1} i \times Pr[x = i]$$

Ayuda: usar la siguiente igualdad

$$\sum_{1=n}^{n} i = \frac{n(n+1)}{2}; \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

(d) Demostrar que para valores grandes de *N*, el número medio de pistas atravesadas en una búsqueda se aproxima a *N*/3.

6.2. Se define lo siguiente para un disco:

ts = tiempo de búsqueda; tiempo medio para posicionar la cabeza sobre una pista
r = velocidad de rotación del disco, en revoluciones por segundo
n = número de bits por sector
N = capacidad de una pista, en bits
t_A = tiempo de acceso a un sector

Desarrollar una fórmula para t_A en función del resto de los parámetros.

6.3. Sea un disco con un solo plato con los siguientes parámetros: velocidad de rotación: 7 200 rpm; número de pistas de una cara del plato: 30 000; número de sectores por pista: 600; tiempo de búsqueda: 1 ms por cada cien pistas atravesadas. El disco recibe una petición de acceso a un sector aleatorio en una pista aleatoria y suponer que la cabeza empieza en la pista 0.

(a) ¿Cuál es el tiempo de búsqueda medio?
(b) ¿Cuál es la latencia rotacional media?
(c) ¿Cuál es el tiempo de transferencia de un sector?
(d) ¿Cuál es el tiempo total medio para realizar una petición?

6.4. Se distingue entre registros físicos y lógicos. Un **registro lógico** es una serie de datos relacionados tratados como una unidad conceptual, independientemente de donde está almacenada la información. Un **registro físico** es una zona contigua de espacio almacenaje que viene definida por las características del dispositivo de almacenamiento y por el sistema operativo. Suponer un disco en el que cada registro físico contiene treinta registros lógicos de 120 bytes. Calcular cuanto espacio en disco (en sectores, pistas y superficies) se necesitarán para almacenar 300 000 registros lógicos si el disco tiene sectores fijos de 512 bytes/sector, con 96 sectores/pista, 110 pistas por superficie, y ocho superficies útiles. Ignorar registros de cabecera de fichero e índices de pista, y suponer que los registros no pueden ocupar dos sectores.

6.5. Debería quedar claro que la organización en tiras de un disco puede mejorar la velocidad de transferencia de datos cuando el tamaño de las tiras es pequeño comparado con el tamaño de la petición de E/S. También debería quedar claro que RAID 0 ofrece mejores prestaciones en lo que se refiere a discos grandes, ya que se pueden gestionar en paralelo varias peticiones de E/S. Sin embargo, en este último caso, ¿es necesaria la organización en tiras del disco? Es decir, ¿la organización en tiras del disco mejora la velocidad de las peticiones de E/S comparada con un conjunto de discos sin tiras?

CAPÍTULO 7

Entrada/salida

ASPECTOS CLAVE

- La arquitectura de E/S del computador es su interfaz con el exterior. Esta arquitectura se diseña de manera que permita una forma sistemática de controlar las interacciones con el mundo exterior y proporcione al sistema operativo la información que necesita para gestionar eficazmente la actividad de E/S.
- Hay tres técnicas de E/S principales: **E/S programada**, en la que la E/S se produce bajo el control directo y continuo del programa que solicita la operación de E/S; **E/S mediante interrupciones**, en la que el programa genera una orden de E/S y después continúa ejecutándose hasta que el hardware de E/S lo interrumpe para indicar que la operación de E/S ha concluido; y **acceso directo a memoria (**DMA, *Direct Memory Access***)**, en el que un procesador de E/S específico toma el control de la operación de E/S para transferir un gran bloque de datos.
- Dos ejemplos importantes de interfaces de E/S externas son **FireWire** e **Infiniband**.

Junto con el procesador y el conjunto de módulos de memoria, el tercer elemento clave de un computador es un conjunto de módulos de E/S. Cada módulo se conecta al bus del sistema o a un conmutador central y controla uno o más dispositivos periféricos. Un módulo de E/S no es únicamente un conector mecánico que permite enchufar el dispositivo al bus del sistema; sino que además está dotado de cierta «inteligencia», es decir, contiene la lógica necesaria para para permitir la comunicación entre el periférico y el bus.

El lector podría preguntarse por qué los periféricos no se conectan directamente al bus del sistema. Las razones son:

- Hay una amplia variedad de periféricos con formas de funcionamiento diferentes. Podría ser imposible incorporar la lógica necesaria dentro del procesador para controlar tal diversidad de dispositivos.
- A menudo la velocidad de transferencia de datos de los periféricos es mucho menor que la de la memoria o el procesador. Así, no es práctico utilizar un bus del sistema de alta velocidad para comunicarse directamente con un periférico.
- Por otro lado, la velocidad de transferencia de algunos periféricos es mayor que la de la memoria o el procesador. De nuevo, esta diferencia daría lugar a comportamientos poco eficientes si no se gestionase correctamente.
- Con frecuencia, los periféricos utilizan datos con formatos y tamaños de palabra diferentes de los del computador a los que se conectan.

En consecuencia, se necesita un módulo de E/S. Este módulo tiene dos funciones principales (Figura 7.1):

- Realizar la interfaz entre el procesador y la memoria a través del bus del sistema o un conmutador central.

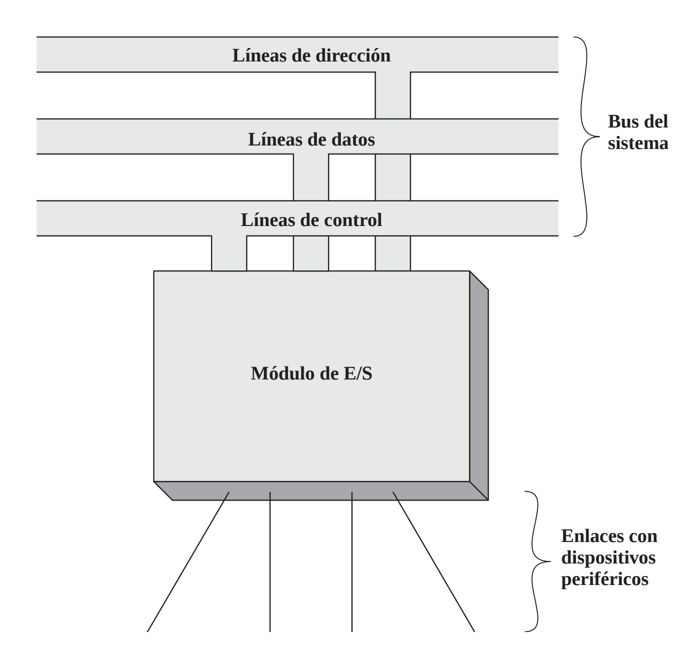


Figura 7.1. Módulo genérico de un módulo E/S.

- Realizar la interfaz entre uno o más dispositivos periféricos mediante enlaces de datos específicos.

Comenzamos este capítulo con una breve discusión acerca de los dispositivos externos, seguida de una revisión de la estructura y el funcionamiento de un módulo de E/S. Después, consideraremos las diferentes formas de realizar la función de E/S en cooperación con el procesador y la memoria: la interfaz de E/S interna. Por último se examina la interfaz de E/S externa, entre el módulo de E/S y el mundo exterior.

7.1. DISPOSITIVOS EXTERNOS

Las operaciones de E/S se realizan a través de una amplia gama de dispositivos que proporcionan una forma de intercambiar datos entre el exterior y el computador. Un dispositivo externo se conecta al computador mediante un enlace a un módulo de E/S (Figura 7.1). El enlace se utiliza para intercambiar señales de control, estado, y datos entre el módulo de E/S y el dispositivo externo. Un dispositivo externo conectado a un módulo de E/S frecuentemente se denomina *dispositivo periférico* o simplemente *periférico*.

En sentido amplio, los dispositivos externos se pueden clasificar en tres categorías:

- *De interacción con humanos:* permiten la comunicación con el usuario del computador.
- *De interacción con máquinas:* permiten la comunicación con elementos del equipo.
- *De comunicación:* permiten la comunicación con dispositivos remotos.

Ejemplos de dispositivos de interacción con humanos son los terminales de video (VDT, *Video Display Terminals*) y las impresoras. Ejemplos de dispositivos de interacción con máquinas son los discos magnéticos y los sistemas de cinta, y los sensores y actuadores, tales como los que se usan en aplicaciones de robótica. Obsérvese que los discos y los sistemas de cinta se están considerando como dispositivos de E/S en este capítulo, mientras que en el Capítulo 6 los consideramos como dispositivos de memoria. Desde el punto de vista de su función, estos dispositivos son parte de la jerarquía de memoria y su uso se discute en el Capítulo 5. Desde un punto de vista estructural, estos dispositivos se controlan mediante módulos de E/S y consecuentemente deben ser considerados en este capítulo.

Los dispositivos de comunicación permiten que el computador intercambie datos con un dispositivo remoto, que puede ser un dispositivo de interacción con humanos, como por ejemplo un terminal, un dispostivo de interacción con máquinas o incluso otro computador.

En términos muy generales, la forma de un dispositivo externo se indica en la Figura 7.2. La conexión con el módulo de E/S se realiza a través de señales de control, estado y datos. Los *datos* se intercambian en forma de un conjunto de bits que son enviados a, o recibidos desde, el módulo de E/S. Las *señales de control* determinan la función que debe realizar el dispositivo, tal como enviar datos al módulo de E/S, ENTRADA («INPUT») o LECTURA («READ»), aceptar datos desde el módulo de E/S, SALIDA («OUTPUT») o ESCRITURA («WRITE»), indicar el estado o realizar alguna función de control particular del dispositivo (por ejemplo, situar una cabeza del disco). Los *datos* son el conjunto de bits a ser enviados o recibidos del módulo de E/S. Las *señales de estado* indican el estado del dispositivo. Como ejemplos están la LISTO/NO-LISTO («READY/NOT-READY») que indica si el dispositivo está preparado para la transferencia de datos.

La *lógica de control* asociada al dispositivo controla su operación en respuesta a las indicaciones del módulo de E/S. El *transductor* convierte las señales eléctricas asociadas al dato a otra forma de energía en el caso de una salida y viceversa en el caso de una entrada. Usualmente, existe un buffer

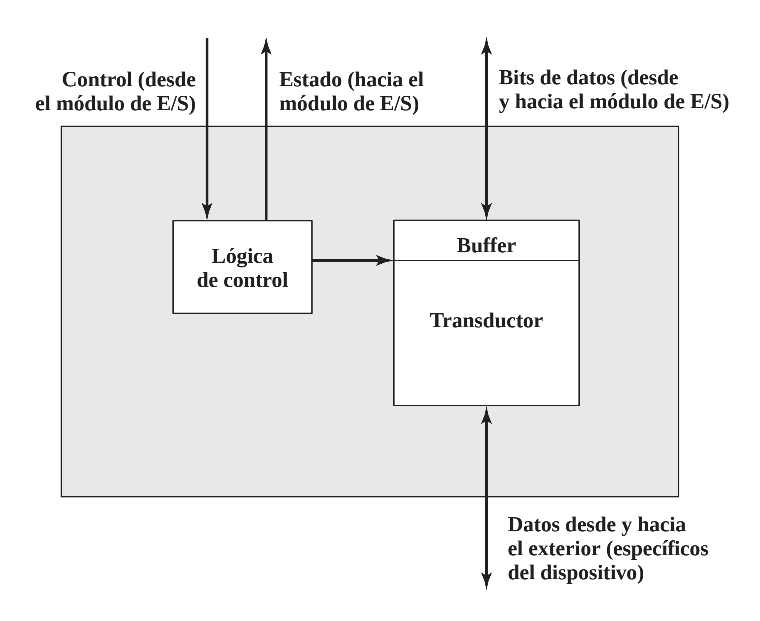


Figura 7.2. Diagrama de bloques de un dispositivo externo.

asociado al transductor para almacenar temporalmente el dato que se está transfiriendo entre el módulo de E/S y el exterior; es común un tamaño de buffer de 8 a 16 bits.

La interfaz entre el módulo de E/S y el dispositivo externo se examinará en la Sección 7.7. La interfaz entre el dispositivo externo y el entorno está fuera del enfoque de este libro, pero se darán algunos ejemplos breves.

TECLADO/MONITOR

La forma más común de interacción computador/usuario se produce a través de la combinación teclado/monitor. El usuario proporciona la entrada a través del teclado. A continuación esta entrada se transmite al computador y puede verse en el monitor. Además, el monitor muestra los datos que proporciona el computador.

La unidad básica de intercambio es el carácter. Asociado con cada carácter hay un código, usualmente de siete u ocho bits de longitud. El código más comúnmente utilizado es el IRA (*International Reference Alphabet*) [1]. Cada carácter de este código se representa mediante un único número binario de 7 bits; en consecuencia, se pueden representar 128 caracteres. La Tabla 7.1 enumera los valores del código. En la tabla, los bits de cada carácter se designan desde b_7, que es el bit más significativo, a b_1, el menos significativo [2]. Los caracteres son de dos tipos: imprimibles y de control (Tabla 7.2). Los caracteres imprimibles son alfabéticos, numéricos y especiales, que pueden imprimirse en papel o visualizarse en una pantalla. Por ejemplo, la representación binaria del carácter *K* es $b_7b_6b_5b_4b_3b_2b_1 = 1001011$. Algunos de los caracteres de control se utilizan para controlar la impresora o la visualización de los caracteres; un ejemplo es el retorno de carro. Otros caracteres de control están relacionados con los procedimientos de comunicación.

Para la entrada desde teclado, cuando el usuario pulsa una tecla, se genera una señal electrónica interpretada por el transductor del teclado que la traduce al patrón binario del correspondiente código IRA. Entonces, este patrón binario se transmite al módulo de E/S del computador. En el computador, el texto se puede almacenar utilizando el mismo código IRA. En la salida, los códigos IRA se transmiten al dispositivo externo desde el módulo de E/S. El transductor del dispositivo interpreta este código y envía las señales electrónicas precisas para que muestre en pantalla el carácter indicado o realice la función de control solicitada.

CONTROLADOR DE DISCO (*DISK DRIVE*)

Un controlador de disco contiene la electrónica necesaria para intercambiar señales de datos, control y estado con un módulo de E/S, más la electrónica para controlar el mecanismo de lectura/escritura del disco. En un disco de cabeza fija, el transductor hace la conversión entre los patrones magnéticos de la superficie del disco en movimiento y los bits del buffer del dispositivo (Figura 7.2). Un disco de

---

[1] El código IRA se define en la recomendación ITU-T T.50 y se conocía inicialmente como Alfabeto Internacional número 5 (International Alphabet Number 5, IA5). La versión de Estados Unidos del IRA se denomina ASCII (American Standard Code for Information Interchange.

[2] Los caracteres IRA casi siempre se almacenan y transmiten utilizando ocho bits por carácter (un bloque de ocho bits se llama **octeto**, o **byte**). El octavo bit es un bit de paridad para la detección de errores. El bit de paridad es el que se encuentra la posición más significativa y por consiguiente se designa como b_8.

Tabla 7.1. El código ASCII (*American Standard Code for Information Interchange*).

bit posición											
b₇				0	0	0	0	1	1	1	1
	b₆			0	0	1	1	0	0	1	1
		b₅		0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1								
0	0	0	0	NUL	DLE	SP	0	@	P	‘	p
0	0	0	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	STX	DC2	“	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	s
0	1	0	0	EOT	DC4	$	4	D	T	d	t
0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	’	7	G	W	g	w
1	0	0	0	BS	CAN	(	8	H	X	h	x
1	0	0	1	HT	EM	)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[	k	{
1	1	0	0	FF	FS	,	<	L	\	l	|
1	1	0	1	CR	GS	–	=	M	]	m	}
1	1	1	0	SO	RS	.	>	N	^	n	~
1	1	1	1	SI	US	/	?	O	_	o	DEL

TABLA 7.2. Caracteres de control IRA.

Control de Formato

BS (*Backspace,* Espacio atrás): indica movimiento de un espacio hacia atrás del mecanismo de impresión o del cursor de la pantalla.

HT (*Horizontal Tab,* Tabulación Horizontal): indica movimiento del mecanismo de impresión o del cursor de pantalla hasta el siguiente «tabulador» asignado o a la posición de parada.

LF (*Line Feed,* Avance de Línea): indica movimiento del mecanismo de impresión o del cursor de pantalla hasta el comienzo de la línea siguiente.

VT (*Vertical Tab,* Tabulación Vertical): indica movimiento del mecanismo de impresión o del cursor de pantalla hasta la siguiente de una serie de líneas impresas.

FF (*Form Feed,* Avance de Página): indica movimiento del mecanismo de impresión o del cursor de pantalla hasta el comienzo de la siguiente página o imagen de pantalla.

CR (*Carriage Return,* Retorno de Carro): indica movimiento del mecanismo de impresión o del cursor de pantalla hasta el comienzo de la línea en curso.

(Continúa)

TABLA 7.2. Caracteres de control IRA (*continuación).*

Control de Transmisión

SOH (*Start of Heading,* Comienzo de Cabecera): usado para indicar el comienzo de una cabecera que puede contener información de dirección o enrutamiento.

STX (*Start of Text,* Comienzo de Texto): usado para indicar el comienzo de texto y también para indicar el final de la cabecera.

ETX (*End of Text,* Final de Texto): usado como fin del texto que se inició con STX.

EOT (*End of Transmission,* Final de Transmisión): indica el final de la transmisión que puede incluir uno o más textos con sus cabeceras.

ENQ (*Enquiry,* Interrogación): petición de respuesta desde una estación remota. Se puede utilizar como una petición de identificación («WHO ARE YOU?») para la estación.

ACK (*Acknowledge,* Reconocimiento): carácter transmitido por un dispositivo receptor como respuesta afirmativa al emisor. Se utiliza como respuesta positiva a los mensajes de sondeo (*polling*).

NAK (*Negative Acknowledgement,* Reconocimiento Negativo): carácter transmitido por un dispositivo receptor como respuesta negativa al emisor. Se utiliza como respuesta negativa a los mensajes de sondeo.

SYN (*Synchronous/Idle,* Síncrono/Parado): utilizado por un sistema con transmisión síncrona para conseguir la sincronización. Cuando no se envía ningún dato un sistema con transmisión síncrona envía caracteres SYN contínuamente.

ETB (*End of Transmission Block,* Final del Bloque Transmitido): se utiliza en el contexto de las comunicaciones para indicar el final de un bloque de datos. Permite organizar los datos en bloques donde la estructura del bloque no está necesariamente relacionada con el formato de procesamiento.

Separadores de Información

FS (*File Separator,* Separador de Fichero)

GS (*Group Separator,* Separador de Grupo)

RS (*Record Separator,* Separador de Registro)

US (*United Separator,* Separador Unido)

Los separadores de información se utilizan de manera opcional si bien están ordenados jerárquicamente desde el FS (el más inclusivo) hasta US (el menos).

Miscelánea

NUL (*Nul,* Nulo): carácter nulo. Se utiliza para consumir tiempo u ocupar espacio en una cinta cuando no hay datos.

BEL (*Bell,* Pitido): utilizado para llamar la atención humana. Puede controlar una alarma o dispositivos que requieren llamar la atención.

SO (*Shift out,* Fuera de Código): indica que los caracteres que siguen no deben interpretarse utilizando el estándar hasta que llegue un carácter SI.

SI (*Shift in,* Dentro de Código): indica que los caracteres que siguen deben interpretarse de acuerdo con el estándar.

DEL (*Delete,* Borrar): utilizado para borrar caracteres (por ejemplo en una cinta de papel perforando cada posición de bit).

SP (*Space,* Espacio): carácter no imprimible utilizado para separar palabras o para mover el mecanismo de impresión, o también para adelantar el cursor una posición.

DLE (*Data Link Escape,* Salir del Enlace de Datos): carácter que puede cambiar el significado de uno o más caracteres consecutivos que lo siguen. Puede proporcionar caracteres de control suplementarios, o permitir el envío de caracteres de datos con cualquier combinación de bits.

(Continúa)

TABLA 7.2. Caracteres de control IRA (*continuación).*

Miscelánea	
DC1, DC2, DC3, DC4 (*Device Controls,* Controles de Dispositivo): caracteres para el control de ciertos dispositivos auxiliares o características especiales de los terminales. **CAN** (*Cancel,* Cancelar): indica que el dato que le precede en el mensaje o en el bloque debe descartarse (normalmente porque se ha detectado un error). **EM** (*End od Medium,* Fin del Medio): indica el final físico de una tarjeta, una cinta u otro medio, o el final de la parte del medio solicitada o utilizada.	**SUB** (*Substitute,* Sustituir): sustituir un caracter que se ha detectado como erróneo o no válido. **ESC** (*Escape,* Salir): carácter pensado para proporcionar una ampliación del código de forma que permite que un número especificado de caracteres contiguos que le siguen tengan un significado alternativo.

cabeza móvil además debe ser capaz de mover radialmente el brazo del disco hacia dentro y hacia fuera sobre la superficie del disco.

7.2. MÓDULOS DE E/S

FUNCIONES DE UN MÓDULO

Las principales funciones y requisitos de un módulo de E/S se encuentran dentro de las siguientes categorías:

- Control y temporización.
- Comunicación con el procesador.
- Comunicación con los dispositivos.
- Almacenamiento temporal de datos.
- Detección de errores.

En cualquier momento, el procesador puede comunicarse con uno o más dispositivos externos en cualquier orden, según las necesidades de E/S del programa. Los recursos internos, tales como la memoria principal y el bus del sistema, deben compartirse entre distintas actividades incluyendo la E/S de datos. Así, la función de E/S incluye ciertos requisitos de **control y temporización**, para coordinar el tráfico entre los recursos internos y los dispositivos externos. Por ejemplo, el control de la transferencia de datos desde un dispositivo externo al procesador podría implicar la siguiente secuencia de pasos:

1. El procesador interroga al módulo de E/S para comprobar el estado del dispositivo conectado al mismo.
2. El módulo de E/S devuelve el estado del dispositivo.

3. Si el dispositivo está operativo y preparado para transmitir, el procesador solicita la transferencia del dato mediante una orden al módulo de E/S.
4. El módulo de E/S obtiene un dato (por ejemplo, de 8 o 16 bits) del dispositivo externo.
5. Los datos se transfieren desde el módulo de E/S al procesador.

Si el sistema utiliza un bus, entonces cada una de las interacciones entre el procesador y el módulo de E/S implican uno o más arbitrajes del bus.

Además, el esquema simplificado previo muestra que el módulo de E/S debe tener la capacidad de entablar comunicación con el procesador y con el dispositivo externo. La **comunicación con el procesador** implica:

- **Decodificación de órdenes:** el módulo de E/S acepta órdenes del procesador. Estas órdenes generalmente se envían utilizando líneas del bus de control. Por ejemplo, un módulo de E/S para un controlador de disco podría recibir las siguientes órdenes: LEER SECTOR («READ SECTOR»), ESCRIBIR SECTOR («WRITE SECTOR»), BUSCAR número de pista (SEEK *track number*), y EXPLORAR IDentificador de registro (SCAN *record ID*). Cada una de las dos últimas órdenes incluye un parámetro que es enviado a través del bus de datos.
- **Datos:** el procesador y el módulo de E/S intercambian datos a través del bus de datos.
- **Información de Estado:** puesto que los periféricos son lentos, es importante conocer el estado del módulo de E/S. Por ejemplo, si se solicita a un módulo de E/S que envíe datos al procesador (lectura), puede que no esté preparado por encontrarse todavía respondiendo a una orden de E/S previa. Esta situación puede indicarse con una señal de estado. Señales de estado usuales son «BUSY» (ocupado) y «READY» (preparado). También puede haber señales para informar de ciertas situaciones de error.
- **Reconocimiento de Dirección:** igual que cada palabra de memoria tiene una dirección, cada dispositivo de E/S tiene otra. Así, un módulo de E/S puede reconocer una única dirección para cada uno de los periféricos que controla.

Por otra parte, el módulo de E/S debe ser capaz de **comunicarse con el dispositivo**. Esta comunicación implica intercambiar órdenes, información del estado, y datos (Figura 7.2).

Una tarea esencial para un módulo de E/S es el **almacenamiento temporal de datos (***data buffering*). La necesidad de esta función es clara si se considera la Figura 2.11. Mientras que la velocidad de transferencia desde, y hacia, la memoria principal o el procesador es bastante alta, dicha velocidad puede ser varios órdenes de magnitud menor para la mayoría de los dispositivos periféricos. Los datos provenientes de la memoria se envían al módulo de E/S en ráfagas rápidas. Los datos se almacenan temporalmente en el módulo de E/S y después se envían al periférico a la velocidad de este. En el sentido contrario, los datos se almacenan para no mantener a la memoria ocupada en una operación de transferencia lenta. Así, el módulo de E/S debe ser capaz de operar a las velocidades tanto del dispositivo como de la memoria. Igualmente, si el dispositivo de E/S trabaja a una velocidad mayor que la memoria, el módulo de E/S se encarga del almacenamiento temporal necesario.

Por último, un módulo de E/S a menudo es responsable de la **detección de errores** y de informar de estos errores al procesador. Una clase de errores son los defectos mecánicos y eléctricos en el funcionamiento del dispositivo (por ejemplo papel atascado, pista de disco en mal estado, etc.). Otra

clase está constituida por los cambios accidentales en los bits al transmitirse desde el dispositivo al módulo de E/S. Para detectar estos errores de transmisión frecuentemente se utiliza algún tipo de código de detección de errores. Un ejemplo sencillo es el uso de un bit de paridad en cada carácter de datos. Por ejemplo, un carácter IRA utiliza siete de los bits de un byte. El octavo bit se asigna de manera que el número total de «unos» en el byte sea par (paridad par) o impar (paridad impar). Cuando se recibe un byte, el módulo de E/S comprueba la paridad para determinar si se ha producido un error.

ESTRUCTURA DE UN MÓDULO DE E/S

La complejidad de los módulos de E/S y el número de dispositivos externos que controlan varían considerablemente. Aquí, únicamente pretendemos realizar una descripción muy general (un dispositivo específico, el Intel 82C55A, se describe en la Sección 7.4). La Figura 7.3 muestra un diagrama de bloques de un módulo de E/S. El módulo se conecta al resto del computador a través de un conjunto de líneas (por ejemplo, líneas del bus del sistema). Los datos que se transfieren a, y desde, el módulo se almacenan temporalmente en uno o más registros de datos. Además, puede haber uno o más registros de estado que proporcionan información del estado presente. Un registro de estado también puede funcionar como un registro de control, para recibir información de control del procesador. La lógica que hay en el módulo interactúa con el procesador a través de una serie de líneas de control. Estas son las que utiliza el procesador para proporcionar las órdenes al módulo de E/S. Algunas de las líneas de control pueden ser utilizadas por el módulo de E/S (por ejemplo, para las señales de arbitraje y estado). El módulo también debe ser capaz de reconocer y generar las direcciones asociadas a los dispositivos que controla. Cada módulo de E/S tiene una dirección única o, si controla más de un dispositivo externo, un conjunto único de direcciones. Por último, el módulo de E/S posee la lógica específica para la interfaz con cada uno de los dispositivos que controla.

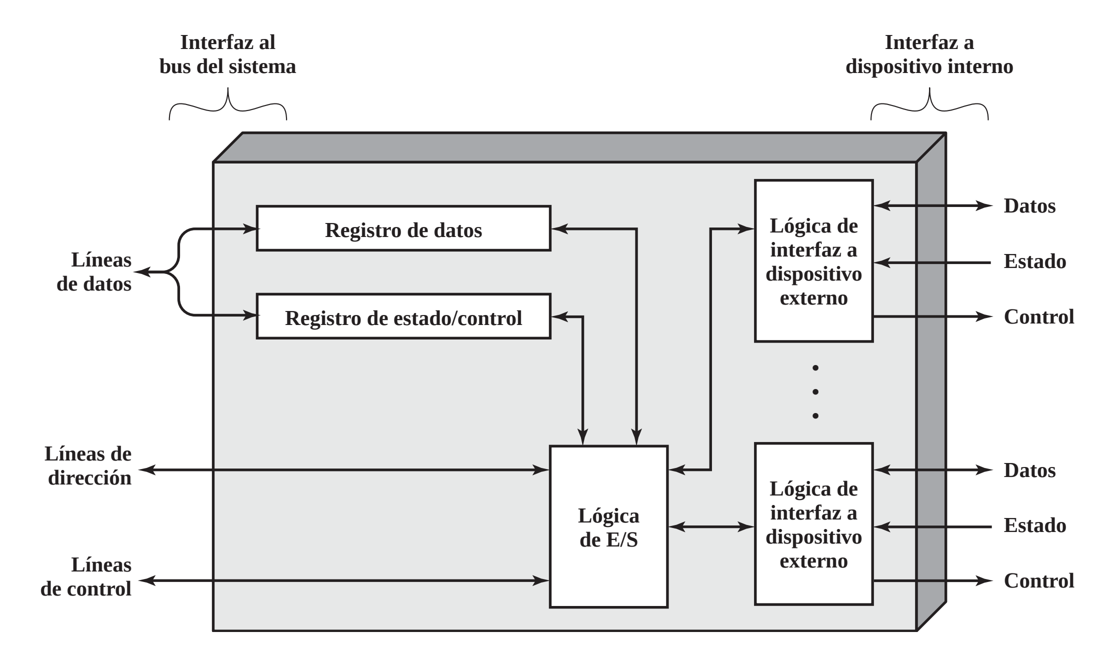


Figura 7.3. Diagrama de bloques de un módulo de E/S.

El funcionamiento de un módulo de E/S permite que el procesador vea a una amplia gama de dispositivos de una forma simplificada. Ante el espectro de posibilidades que pueden darse, el módulo de E/S debe ocultar los detalles de temporización formatos, y electromecánica de los dispositivos externos para que el procesador pueda funcionar únicamente en términos de órdenes de lectura y escritura, y posiblemente órdenes de abrir y cerrar ficheros. En su forma más sencilla, el módulo de E/S puede, no obstante, dejar al procesador parte del trabajo de control del dispositivo (por ejemplo, rebobinar una cinta).

Un módulo de E/S que se encarga de la mayoría de los detalles del procesamiento, presentando al procesador una interfaz de alto nivel, se denomina generalmente *canal de E/S* o *procesador de E/S*. Un módulo que sea bastante simple y requiera un control detallado normalmente se denomina *controlador de E/S* o *controlador de dispositivo*. Los controladores de E/S usualmente aparecen en microcomputadores, mientras que los canales de E/S se utilizan en grandes computadores centrales (*mainframes*).

En lo que sigue, haremos uso del término genérico *módulo de E/S* cuando no haya posibilidad de confusión y utilizaremos los términos específicos cuando sea preciso.

7.3. E/S PROGRAMADA

Son posibles tres técnicas para las operaciones de E/S. Con la *E/S programada*, los datos se intercambian entre el procesador y el módulo de E/S. El procesador ejecuta un programa que controla directamente la operación de E/S, incluyendo la comprobación del estado del dispositivo, el envío de una orden de lectura o escritura y la transferencia del dato. Cuando el procesador envía una orden al módulo de E/S, debe esperar hasta que la operación de E/S concluya. Si el procesador es más rápido que el módulo de E/S, el procesador desperdicia este tiempo. Con la *E/S mediante interrupciones*, el procesador proporciona la orden de E/S, continúa ejecutando otras instrucciones y es interrumpido por el módulo de E/S cuando este ha terminado su trabajo. Tanto con E/S programada como con interrupciones, el procesador es responsable de extraer los datos de la memoria principal en una salida y de almacenar los datos en la memoria principal en una entrada. La alternativa se conoce como *acceso directo a memoria (DMA)*. En este caso, el módulo de E/S y la memoria principal intercambian datos directamente, sin la intervención del procesador.

La Tabla 7.3 indica la relación entre estas tres técnicas. En esta sección, estudiamos la E/S programada. La E/S mediante interrupciones y el DMA se consideran, respectivamente, en las dos próximas secciones.

Tabla 7.3. Técnicas de E/S.

	Sin interrupciones	**Usando interrupciones**
Transferencia de E/S a memoria a través de la CPU	E/S Programada	E/S mediante interrupciones
Transferencia directa de E/S a memoria		Acceso Directo a Memoria (DMA)

RESUMEN DE LA E/S PROGRAMADA

Cuando el procesador está ejecutando un programa y encuentra una instrucción relacionada con una E/S, ejecuta dicha instrucción mandando una orden al módulo de E/S apropiado. Con E/S programada, el módulo de E/S realizará la acción solicitada y después activará los bits apropiados en el registro de estado de E/S (Figura 7.3). El módulo de E/S no realiza ninguna otra acción para avisar al procesador. En concreto, no interrumpe al procesador. De esta forma, el procesador es responsable de comprobar periódicamente el estado del módulo de E/S hasta que encuentra que la operación ha terminado.

Para explicar la técnica de la E/S programada, la consideraremos primero desde el punto de vista de las órdenes de E/S que envía la CPU al módulo de E/S, y después desde el punto de vista de las instrucciones de E/S que ejecuta el procesador.

ÓRDENES DE E/S

Al ejecutar una instrucción relacionada con una E/S, el procesador proporciona una dirección, especificando el módulo de E/S particular y el dispositivo externo, y una orden de E/S. Hay cuatro tipos de órdenes de E/S que puede recibir un módulo de E/S cuando es direccionado por el procesador:

- **Control:** se utiliza para activar el periférico e indicarle qué hacer. Por ejemplo, puede indicarse a una unidad de cinta magnética que se rebobine o que avance al registro siguiente. Estas órdenes son específicas del tipo particular de periférico.
- **Test:** se utiliza para comprobar diversas condiciones de estado asociadas con el módulo de E/S y sus periféricos. El procesador podrá comprobar si el periférico en cuestión está conectado y disponible para su uso. También podrá saber si la operación de E/S más reciente ha terminado y si se ha producido algún error.
- **Lectura:** hace que el módulo de E/S capte un dato de un periférico y lo sitúe en un buffer interno (representado como un registro de datos en la Figura 7.3). Después, el procesador puede obtener el dato solicitando que el módulo de E/S lo ponga en el bus de datos.
- **Escritura:** hace que el módulo de E/S capte un dato (byte o palabra) del bus de datos y posteriormente lo transmita al periférico.

La Figura 7.4a proporciona un ejemplo del uso de la E/S programada para leer un bloque de datos desde un dispositivo periférico (por ejemplo, un registro de una cinta) y almacenarlo en memoria. Los datos se leen palabra a palabra (16 bits, por ejemplo). Por cada palabra leida, el procesador debe permanecer en un ciclo de comprobación de estado hasta que determine que la palabra está disponible en el registro de datos del módulo de E/S. Este diagrama de flujo resalta la principal desventaja de esta técnica: es un proceso que consume tiempo y mantiene al procesador innecesariamente ocupado.

INSTRUCCIONES DE E/S

En la E/S programada, hay una estrecha correspondencia entre las instrucciones de E/S que el procesador capta de memoria y las órdenes de E/S que el procesador envía a un módulo de E/S al ejecutar

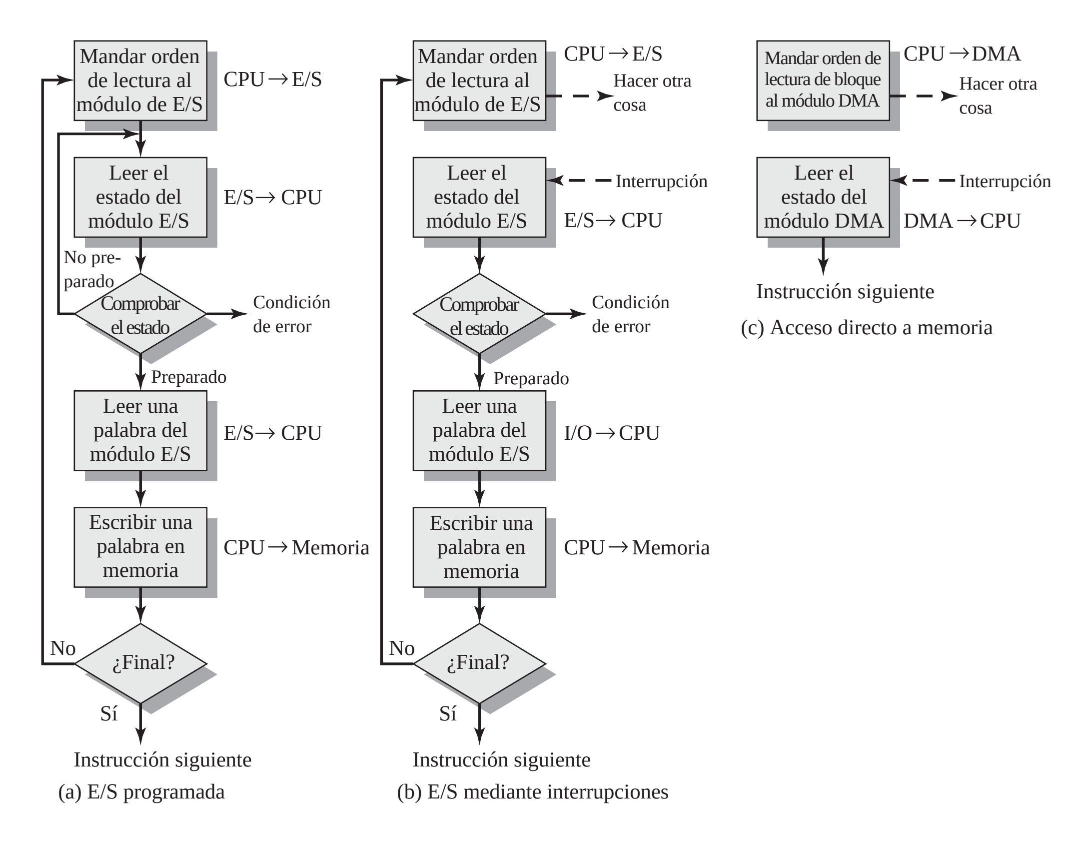


Figura 7.4. Tres técnicas para la entrada de un bloque de datos.

las instrucciones. Es decir, las instrucciones se pueden hacer corresponder fácilmente con las órdenes de E/S, y a menudo hay una simple relación de uno a uno. La forma de la instrucción depende de la manera de direccionar los dispositivos externos.

Normalmente, habrá muchos dispositivos de E/S conectados al sistema a través de los módulos de E/S. Cada dispositivo tiene asociado un identificador único o dirección. Cuando el procesador envía una orden de E/S, la orden contiene la dirección del dispositivo deseado. Así, cada módulo de E/S debe interpretar las líneas de dirección para determinar si la orden es para él.

Cuando el procesador, la memoria principal, y las E/S comparten un bus común, son posibles dos modos de direccionamiento: asignado en memoria (*memory-mapped*) y aislado. Con las **E/S asignadas en memoria**, existe un único espacio de direcciones para las posiciones de memoria y los dispositivos de E/S. El procesador considera a los registros de estado y de datos de los módulos de E/S como posiciones de memoria y utiliza las mismas instrucciones máquina para acceder tanto a memoria como a los dispositivos de E/S. Así, por ejemplo, con diez líneas de dirección, se puede acceder a un total de $2^{10} = 1024$ posiciones de memoria y direcciones de E/S, en cualquier combinación.

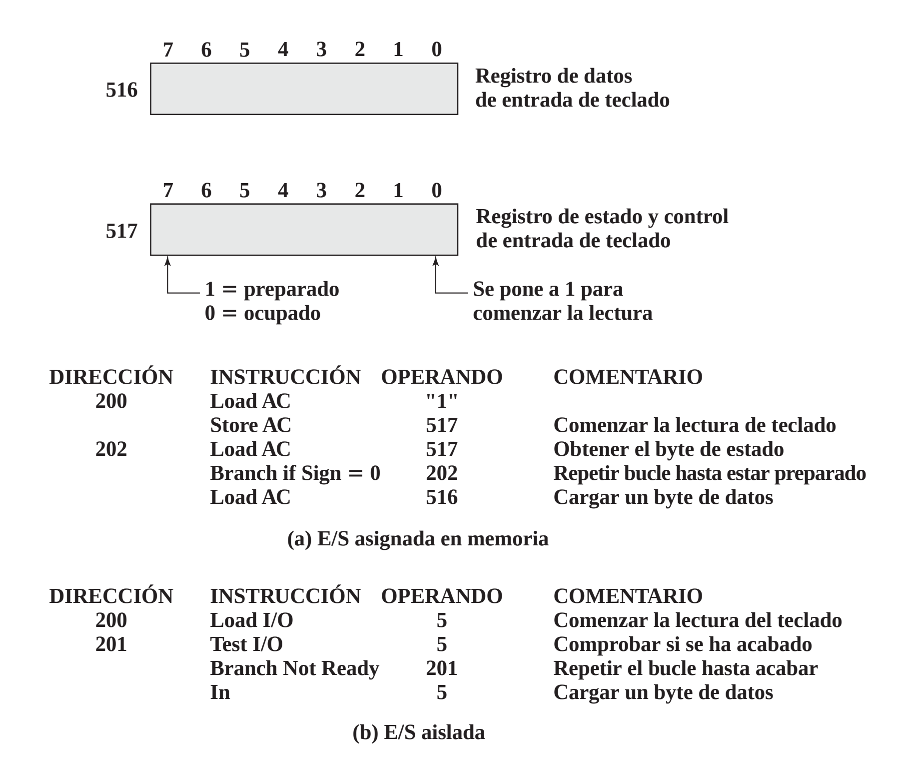
Con las E/S asignadas en memoria, se necesita una sola línea de lectura y una sola línea de escritura en el bus. Alternativamente, el bus puede disponer de líneas de lectura y escritura en memoria junto con líneas para órdenes de entrada y salida. En este caso, las líneas de órdenes especifican si la dirección se refiere a una posición de memoria o a un dispositivo de E/S. El rango completo de

direcciones está disponible para ambos. De nuevo, con diez líneas de dirección, el sistema puede soportar ahora 1 024 posiciones de memoria y 1 024 direcciones de E/S. Puesto que el espacio de direcciones de E/S está aislado del de memoria, éste se conoce con el nombre de **E/S aislada**.

En la Figura 7.5 se contrastan estas dos técnicas de E/S programada. La Figura 7.5a muestra cómo podría ver el programador la interfaz con un dispositivo de entrada sencillo, tal como un teclado, cuando se utiliza E/S asignada en memoria. Se asumen direcciones de diez bits, con una memoria de 512 palabras (posiciones 0-511) y hasta 512 direcciones de E/S (posiciones 512-1023). Se dedican dos direcciones a la entrada de teclado desde un terminal concreto. La dirección 516 se refiere al registro de datos y la dirección 517 al registro de estado, que además funciona como registro de control para recibir las órdenes del procesador. El programa que se muestra lee un byte de datos desde el teclado y lo escribe en el registro acumulador del procesador. Obsérvese cómo el procesador ejecuta un bucle hasta que el byte de datos está disponible.

Con E/S aislada (Figura 7.5b), los puertos de E/S solo son accesibles mediante una orden específica de E/S, que activa las líneas de órdenes de E/S del bus.

La mayor parte de procesadores disponen de un conjunto relativamente grande de instrucciones distintas para acceder a memoria. Si se utiliza E/S aislada, solo existen unas pocas instrucciones de E/S. Por eso, una ventaja de la E/S asignada en memoria es que se puede utilizar este amplio repertorio de instrucciones, permitiendo una programación más eficiente. Una desventaja es que se utiliza parte del valioso espacio de direcciones de memoria. Tanto la E/S asignada en memoria como la aislada se usan comúnmente.



DIRECCIÓN	INSTRUCCIÓN	OPERANDO	COMENTARIO
200	Load AC	"1"	
	Store AC	517	Comenzar la lectura de teclado
202	Load AC	517	Obtener el byte de estado
	Branch if Sign = 0	202	Repetir bucle hasta estar preparado
	Load AC	516	Cargar un byte de datos

(a) E/S asignada en memoria

DIRECCIÓN	INSTRUCCIÓN	OPERANDO	COMENTARIO
200	Load I/O	5	Comenzar la lectura del teclado
201	Test I/O	5	Comprobar si se ha acabado
	Branch Not Ready	201	Repetir el bucle hasta acabar
	In	5	Cargar un byte de datos

(b) E/S aislada

Figura 7.5. E/S mapeada en memoria y mapa de E/S aislado.

7.4. E/S MEDIANTE INTERRUPCIONES

El problema con la E/S programada es que el procesador tiene que esperar un tiempo considerable a que el módulo de E/S en cuestión esté preparado para recibir o transmitir los datos. El procesador, mientras espera, debe comprobar repetidamente el estado del módulo de E/S. Como consecuencia, se degrada el nivel de prestaciones de todo el sistema.

Una alternativa consiste en que el procesador, tras enviar una orden de E/S a un módulo, continúe realizando algún trabajo útil. Después, el módulo de E/S interrumpirá al procesador para solicitar su servicio cuando esté preparado para intercambiar datos con él. El procesador ejecuta entonces la transferencia de datos, como antes, y después continúa con el procesamiento previo.

Estudiemos cómo funciona, primero desde el punto de vista del módulo de E/S. Para una entrada, el módulo de E/S recibe una orden READ del procesador. Entonces, el módulo de E/S procede a leer el dato desde el periférico asociado. Una vez que el dato está en el registro de datos del módulo, el módulo envía una interrupción al procesador a través de una línea de control. Después, el módulo espera hasta que el procesador solicite su dato. Cuando ha recibido la solicitud, el módulo sitúa su dato en el bus de datos y pasa a estar preparado para otra operación de E/S.

Desde el punto de vista del procesador, las acciones para una entrada son las que siguen. El procesador envía una orden READ de lectura. Entonces pasa a realizar otro trabajo (es decir, el procesador puede estar ejecutando programas distintos al mismo tiempo). Al final de cada ciclo de instrucción, el procesador comprueba las interrupciones (Figura 3.9). Cuando se pide la interrupción desde el módulo de E/S, el procesador guarda el contexto (es decir, el contador de programa y los registros del procesador) del programa en curso y procesa la interrupción. En este caso, el procesador lee la palabra de datos del módulo de E/S y la almacena en memoria. Después recupera el contexto del programa que estaba ejecutando (o de otro programa) y continúa su ejecución.

La Figura 7.4b muestra el uso de E/S con interrupciones para leer un bloque de datos. Compárese con la Figura 7.4a. La E/S con interrupciones es más eficiente que la E/S programada porque elimina las esperas innecesarias. No obstante, las E/S con interrupciones todavía consumen gran cantidad del tiempo del procesador puesto que cada palabra de datos que va desde la memoria al módulo de E/S o viceversa debe pasar a través del procesador.

PROCESAMIENTO DE LA INTERRUPCIÓN

Consideremos con más detalle el papel del procesador en las E/S. Cuando se produce una interrupción se disparan una serie de eventos en el procesador, tanto a nivel hardware como software. La Figura 7.6 muestra una secuencia típica. Cuando un dispositivo de E/S termina una operación de E/S, se produce la siguiente secuencia de eventos en el hardware:

1. El dispositivo envía una señal de interrupción al procesador.
2. El procesador termina la ejecución de la instrucción en curso antes de responder a la interrupción, como indica la Figura 3.9.
3. El procesador comprueba si hay interrupciones, determina que hay una, y envía una señal de reconocimiento al dispositivo que originó la interrupción. La señal de reconocimiento hace que el dispositivo desactive su señal de interrupción.

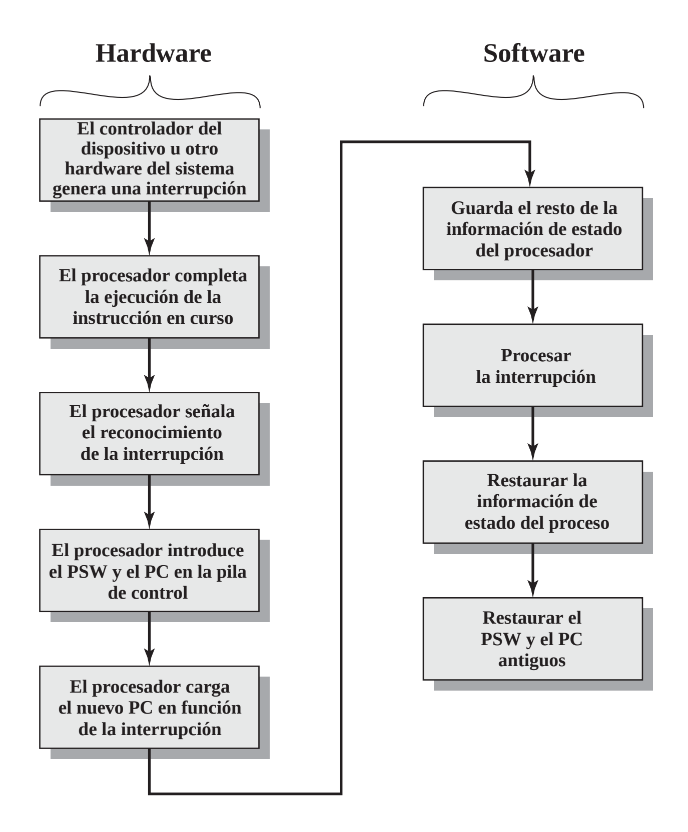


Figura 7.6. Procedimiento de interrupción simple.

4. Ahora el procesador necesita prepararse para transferir el control a la rutina de interrupción. Para empezar, debe guardar la información necesaria para continuar el programa en curso en el punto en que se interrumpió. La información mínima que se precisa es (a) el estado del procesador, que se almacena en un registro llamado Palabra de Estado del Programa (PSW, *Program Status Word*), y (b) la posición de la siguiente instrucción a ejecutar, que está contenida en el contador de programa. Estos registros se pueden introducir en la pila de control del sistema [3].

5. Después, el procesador carga el contador de programa con la posición de inicio del programa de gestión de la interrupción solicitada. Según sea la arquitectura del computador y el diseño del sistema operativo, puede haber un solo programa, uno por cada tipo de interrupción, o uno por cada dispositivo y cada tipo de interrupción. Si hay más de una rutina de gestión de interrupción, el procesador debe determinar a qué programa llamar. Esta información puede haber sido incluida en la señal de interrupción original, o el procesador puede tener que enviar una solicitud al dispositivo que originó la interrupción para que este responda con la información que se precise.

Una vez que el contador de programa se ha cargado, el procesador continúa con el ciclo de instrucción siguiente, que empieza con la captación de instrucción. Puesto que la instrucción a captar

[3] *Véase* en el Apéndice 10A un análisis sobre el funcionamiento de la pila.

viene determinada por el contenido del contador de programa, el control se transfiere al programa de gestión de interrupción. La ejecución de este programa da lugar a las siguientes operaciones:

6. Hasta este momento, se han almacenado en la pila del sistema el contador de programa y el PSW del programa interrumpido. Sin embargo, hay otra información que se considera parte del «estado» de un programa en ejecución. En concreto, se deben guardar los contenidos de los registros del procesador puesto que estos registros pueden ser utilizados por la rutina de interrupción. Usualmente, la rutina de gestión de interrupción empezará almacenando en la pila los contenidos de todos los registros. La Figura 7.7a muestra un ejemplo sencillo. En este caso, un programa de usuario es interrumpido después de la instrucción de la posición N. Los contenidos de todos los registros junto con la dirección de la siguiente instrucción ($N + 1$) se introducen en la pila. El puntero de la pila se actualiza para que apunte a la nueva cabecera de la pila, y el contador de programa se actualiza para que apunte al comienzo de la rutina de servicio de interrupción.

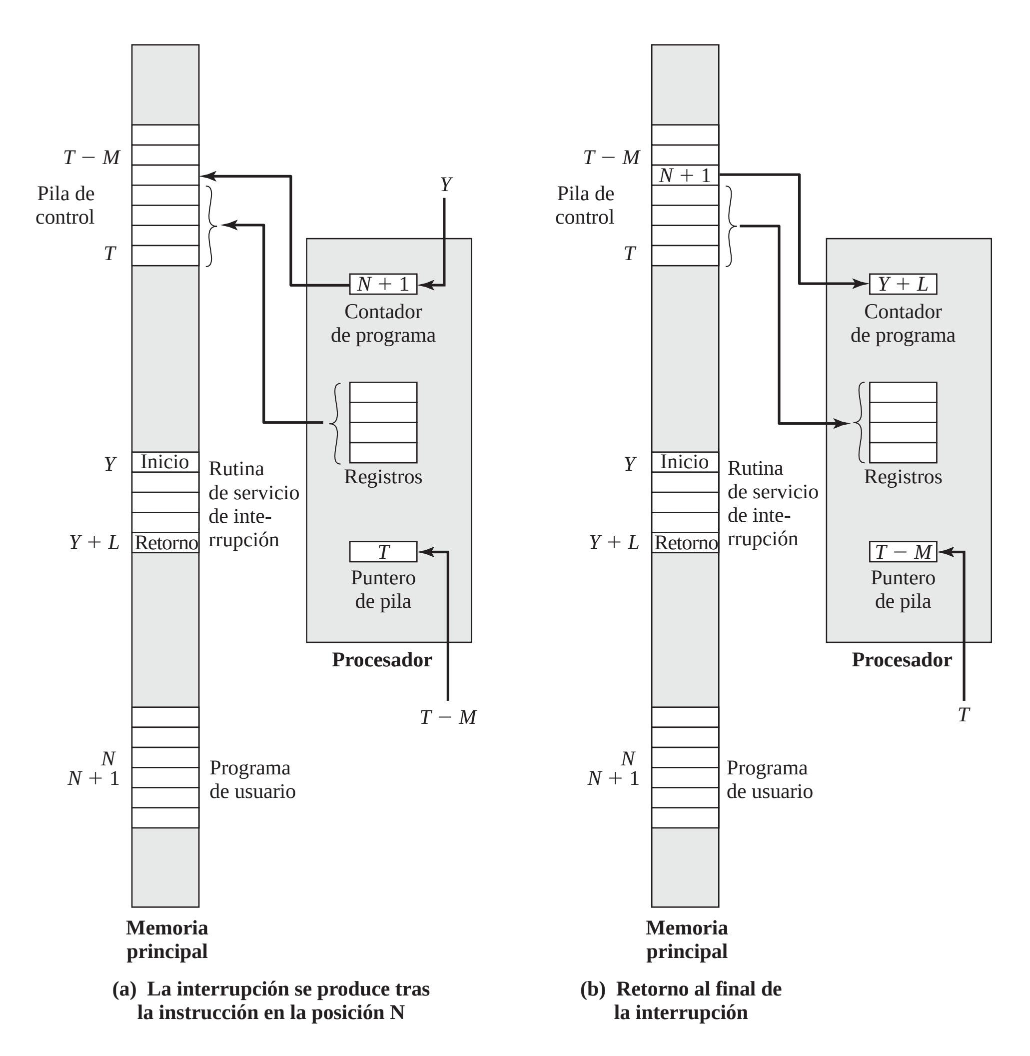


Figura 7.7. Cambios en memoria y en los registros debido a una interrupción.

7. La rutina de gestión de la interrupción puede continuar ahora procesando la interrupción. Esto incluirá el examen de la información de estado relativa a la operación de E/S o a cualquier otro evento que causara la interrupción. También puede implicar el envío al dispositivo de E/S de órdenes o señales de reconocimiento adicionales.
8. Cuando el procesamiento de la interrupción ha terminado, los valores de los registros almacenados se recuperan de la pila y se vuelven a almacenar en los registros (por ejemplo, *véase* la Figura 7.7b).
9. El paso final es recuperar los valores del PSW y del contador de programa desde la pila. Como resultado, la siguiente instrucción que se ejecute pertenecerá al programa previamente interrumpido.

Obsérvese que es importante almacenar toda la información del estado del programa interrumpido para que este pueda reanudarse. Esto se debe a que la interrupción no es una llamada a una rutina realizada desde el programa. En cambio, la interrupción puede producirse en cualquier momento y por consiguiente en cualquier punto de la ejecución del programa de usuario. Una interrupción es impredecible. De hecho, como se verá en el siguiente capítulo, los dos programas pueden no tener nada en común y pueden pertenecer a distintos usuarios.

CUESTIONES DE DISEÑO

En la implementación de las E/S mediante interrupciones aparecen dos cuestiones. Primero, puesto que casi invariablemente habrá múltiples módulos de E/S, cómo determina el procesador qué dispositivo ha provocado la interrupción. Y segundo, si se han producido varias interrupciones, cómo decide el procesador la que debe atender.

Consideremos en primer lugar la identificación del dispositivo. Hay cuatro tipos de técnicas que se utilizan comúnmente:

- Múltiples líneas de interrupción.
- Consulta software (*software poll*).
- Conexión en cadena, (*Daisy chain*), (Consulta hardware, vectorizada).
- Arbitraje de bus (vectorizada).

La aproximación más directa al problema consiste en proporcionar **varias líneas de interrupción** entre el procesador y los módulos de E/S. Sin embargo, no resulta práctico dedicar más de unas pocas líneas del bus o terminales del procesador a ser líneas de interrupción. En consecuencia, incluso si se utilizan varias líneas, es probable que a cada una se conecten varios módulos de E/S. Por eso, se debe utilizar alguna de las otras tres técnicas en cada línea.

Una alternativa es la **consulta software**. Cuando el procesador detecta una interrupción, se produce una bifurcación a una rutina de servicio de interrupción que se encarga de consultar a cada módulo de E/S para determinar el módulo que ha provocado la interrupción. La consulta podría realizarse mediante una línea específica (por ejemplo, «TEST_E/S»). En este caso, el procesador activa «TEST_E/S» y sitúa la dirección de un módulo de E/S en las líneas de dirección. El módulo de E/S responde positivamente si solicitó la interrupción. Como alternativa, cada módulo de E/S podría

disponer de un registro de estado direccionable. Entonces, el procesador lee el estado del registro de cada módulo de E/S para identificar el módulo que solicitó la interrupción. Una vez identificado el módulo, se produce una bifurcación para que el procesador ejecute la rutina de servicio específica para ese dispositivo.

La desventaja de la consulta software está en el tiempo que consume. Una técnica más eficiente consiste en utilizar la *conexión en cadena (daisy chain)* de los módulos de E/S que proporciona, de hecho, una consulta hardware. Un ejemplo de configuración que utiliza esta conexión en cadena se muestra en la Figura 3.26. Todos los módulos de E/S comparten una línea común para solicitar interrupciones. La línea de reconocimiento de interrupción se conecta encadenando los módulos uno tras otro. Cuando el procesador recibe una interrupción, activa el reconocimiento de interrupción. Esta señal se propaga a través de la secuencia de módulos de E/S hasta que alcanza un módulo que solicitó interrupción. Normalmente este módulo responde colocando una palabra en las líneas de datos. Esta palabra se denomina *vector* y es la dirección del módulo de E/S o algún otro tipo de identificador específico. En cualquier caso, el procesador utiliza el vector como un puntero a la rutina de servicio de dispositivo apropiada. Así se evita tener que ejecutar una rutina de servicio general en primer lugar. Esta técnica se conoce con el nombre de *interrupción vectorizada*.

Hay otra técnica que hace uso de las interrupciones vectorizadas, y se trata de el *arbitraje de bus*. Con el arbitraje de bus, un módulo de E/S debe en primer lugar disponer del control del bus antes de poder activar la línea de petición de interrupción. Así, solo un módulo puede activar la línea en un instante. Cuando el procesador detecta la interrupción, responde mediante la línea de reconocimiento de interrupción. Después, el módulo que solicitó la interrupción sitúa su vector en las líneas de datos.

Las técnicas enumeradas arriba sirven para identificar el módulo de E/S que solicita interrupción. Además proporcionan una forma de asignar prioridades cuando más de un dispositivo está pidiendo que se sirva su interrupción. Con varias líneas de interrupción, el procesador simplemente selecciona la línea con más prioridad. Con la consulta software, el orden en el que se consultan los módulos determina su prioridad. De igual forma, el orden de los módulos en la conexión en cadena (*daisy chain*) determina su prioridad. Finalmente, el arbitraje de bus puede emplear un esquema de prioridad como el discutido en la Sección 3.4.

Ahora pasamos a considerar dos ejemplos de estructuras de interrupción.

CONTROLADOR DE INTERRUPCIONES INTEL 82C59A

El 80386 de Intel posee una sola línea de Petición de Interrupción (INTR, *Interrupt request*) y una sola línea de Reconocimiento de Interrupción (INTA, *Interrupt Acknowledge*). Para que el 80386 pueda manejar flexiblemente cierta variedad de dispositivos y estructuras de prioridad, normalmente se configura con un árbitro de interrupciones externo, el 82C59A. Los dispositivos externos se conectan al 82C59A, que a su vez se conecta al 80386.

La Figura 7.8 ilustra el uso del 82C59A para conectar varios módulos de E/S con el 80386. Un único 82C59A puede manejar hasta ocho módulos. Si se precisa controlar más de ocho módulos, se pueden disponer en cascada para manejar hasta 64 módulos.

La única responsabilidad del 82C59A es la gestión de interrupciones. Acepta las solicitudes de interrupción de los dispositivos conectados a él, determina qué interrupción tiene la prioridad más alta, y se lo indica entonces al procesador activando la señal INTR. El procesador reconoce la

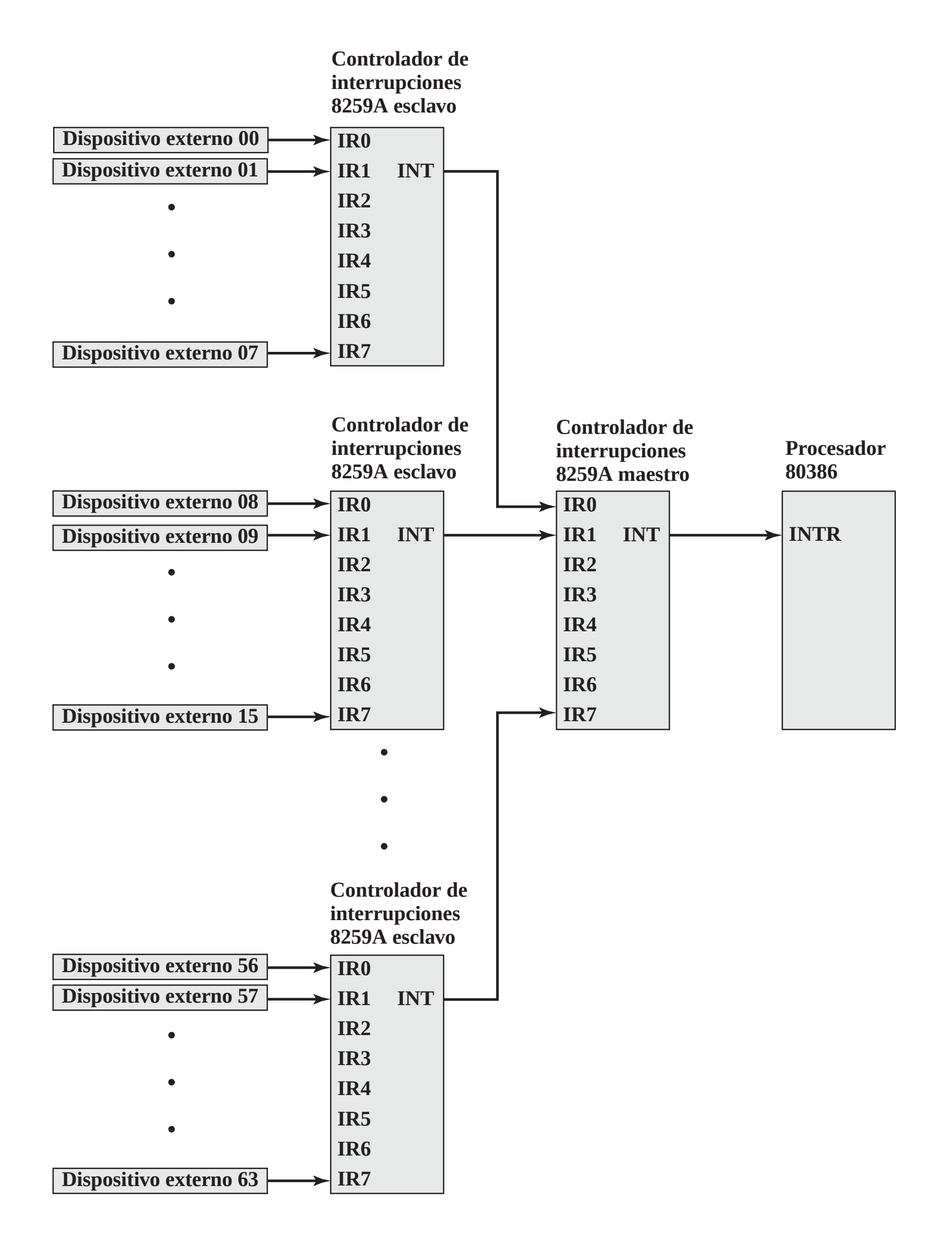


Figura 7.8. Uso del controlador de interrupciones 82C59A.

solicitud mediante la línea INTA. Esto hace que el 82C59A sitúe el vector apropiado en el bus de datos. Entonces, el procesador puede iniciar el procesamiento de la interrupción y comunicarse directamente con el módulo de E/S para leer o escribir datos.

El 82C59A es programable. El 80386 determina el esquema de prioridad que se va a utilizar cargando una palabra de control en el 82C59A. Son posibles los siguientes modos de interrupción.

- **Completamente anidado:** las solicitudes de interrupción se ordenan según un nivel de prioridad desde 0 (IR0) hasta 7 (IR7).

- **Rotatorio:** en algunas aplicaciones hay varios dispositivos con igual prioridad de interrupción. En este modo, un dispositivo pasa a tener la menor prioridad del grupo después de ser servido.
- **Con máscara especial:** se permite que el procesador pueda inhibir selectivamente las interrupciones desde ciertos dispositivos.

LA INTERFAZ PROGRAMABLE DE PERIFÉRICOS INTEL 82C55A

Como ejemplo de un módulo de E/S utilizado para la E/S programada y para la E/S mediante interrupciones, consideramos la interfaz programable de periféricos Intel 82C55A. El 82C55A es un módulo de E/S de propósito general integrado en un solo chip y diseñado para ser usado con el procesador Intel 8086. La Figura 7.9 muestra el diagrama general de bloques junto con la asignación de terminales para el encapsulado de cuarenta terminales que lo contiene.

El lado derecho del diagrama de bloques es la interfaz externa del 82C55A. Las 24 líneas de E/S son programables por el 80386 mediante un registro de control. El 80386 puede fijar el valor del registro de control para especificar los diversos modos de operación y configuraciones. Las 24 líneas se dividen en tres grupos de ocho bits (A, B, C). Cada grupo puede funcionar como un puerto de E/S de ocho bits. Además, el grupo C se subdivide en grupos de cuatro bits (C_A y C_B), que pueden usarse conjuntamente con los puertos de E/S A y B. Configurado de esta forma, esos grupos contienen las señales de control y estado.

El lado izquierdo del diagrama de bloques es la interfaz interna con el bus del 80386. Esta incluye un bus de datos bidireccional de ocho bits (D0 a D7), usado para transferir datos a y desde los puertos de E/S y para transferir la información al registro de control. Las dos líneas de direcciones especifican uno de los tres puertos de E/S o el registro de control. Una transferencia se producirá

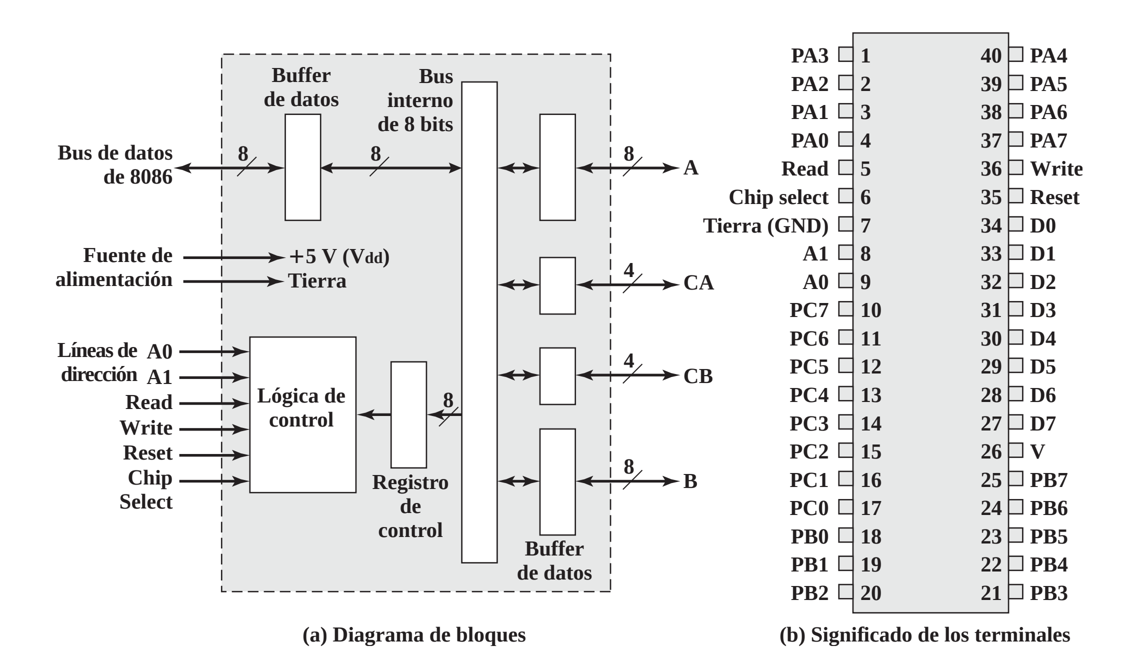


Figura 7.9. Interfaz programable de periféricos 82C55A de Intel.

cuando la línea de selección de chip (CHIP SELECT) se activa junto con la línea de lectura (READ) o escritura (WRITE). La línea RESET se utiliza para iniciar el módulo.

El procesador escribe en el registro de control para seleccionar el modo de operación y para definir las señales, en su caso. En el Modo 0 de operación, los tres grupos de ocho líneas externas funcionan como tres puertos de E/S de ocho bits. Cada puerto puede ser designado como de entrada o de salida. En caso contrario, los grupos A y B funcionan como puertos de E/S, y las líneas del grupo C sirven de líneas de control para A y B. Las líneas de control tienen dos funciones principales: la sincronización mediante conformidad de señales (*handshaking*) y la petición de interrupciones. La conformidad es un mecanismo sencillo de temporización. El emisor utiliza una línea de control como línea de datos listos (DATA READY) para indicar que hay un dato en las líneas de datos de E/S. El receptor utiliza otra línea como reconocimiento (ACKNOWLEDGE), para indicar que el dato se ha leido y que las líneas de datos se pueden liberar. Se puede designar otra línea como línea de Petición de Interrupción (INTERRUPT REQUEST) en el bus del sistema.

Como el 82C55A es programable a través del registro de control, puede utilizarse para controlar diversos dispositivos periféricos simples. La Figura 7.10 ilustra su uso para controlar un terminal con teclado y pantalla. El teclado proporciona ocho bits de entrada. Dos de estos bits, SHIFT y CONTROL, tienen un significado especial para el programa de gestión de teclado que ejecuta el procesador. Sin embargo, este significado es transparente para el 82C55A, que simplemente acepta los

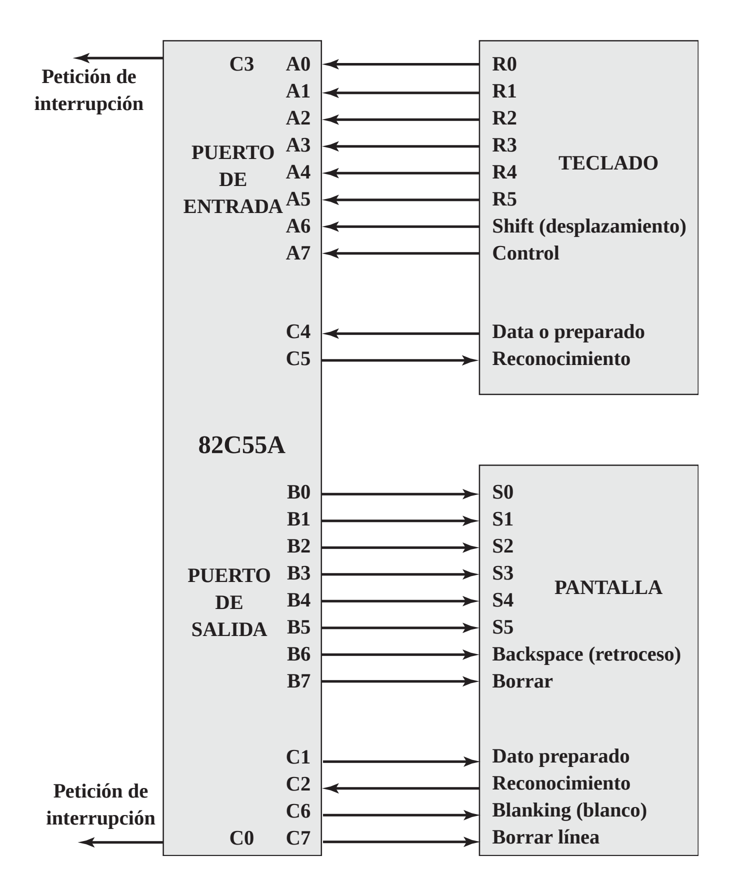


Figura 7.10. Interfaz teclado/pantalla en el 82C55A.

ocho bits de datos y los pone en el bus de datos del sistema. Existen dos líneas para la sincronización del teclado mediante conformidad (*handshaking*).

La pantalla también está conectada a un puerto de datos de ocho bits. De nuevo, dos de los bits tienen un significado específico que es transparente para el 82C55A. Junto a las dos líneas para la sincronización mediante conformidad, hay dos líneas más para funciones de control adicionales.

7.5. ACCESO DIRECTO A MEMORIA

INCONVENIENTES DE LA E/S PROGRAMADA Y CON INTERRUPCIONES

La E/S con interrupciones, aunque más eficiente que la sencilla E/S programada, también requiere la intervención activa del procesador para transferir datos entre la memoria y el módulo de E/S, y cualquier transferencia de datos debe seguir un camino a través del procesador. Por tanto, ambas formas de E/S presentan dos inconvenientes inherentes:

1. La velocidad de transferencia de E/S está limitada por la velocidad a la cual el procesador puede comprobar y dar servicio a un dispositivo.
2. El procesador debe dedicarse a la gestión de las transferencias de E/S; se debe ejecutar cierto número de instrucciones por cada transferencia de E/S (*véase* Figura 7.5).

Existe un cierto compromiso entre estos dos inconvenientes. Considérese una transferencia de un bloque de datos. Utilizando E/S programada, el procesador se dedica a la tarea de la E/S y puede transferir datos a alta velocidad al precio de no hacer nada más. La E/S con interrupciones libera en parte al procesador, a expensas de reducir la velocidad de E/S. No obstante, ambos métodos tienen un impacto negativo tanto en la actividad del procesador como en la velocidad de transferencia de E/S.

Cuando hay que transferir grandes volúmenes de datos, se requiere una técnica más eficiente: el acceso directo a memoria (DMA).

FUNCIONAMIENTO DEL DMA

El DMA requiere un módulo adicional en el bus del sistema. El módulo o controlador de DMA (Figura 7.11) es capaz de imitar al procesador y, de hecho, de recibir el control del sistema cedido por el procesador. Necesita dicho control para transferir datos a, y desde, memoria a través del bus del sistema. Para hacerlo, el módulo de DMA debe utilizar sólo cuando el procesador no lo necesita, o debe forzar al procesador a que suspenda temporalmente su funcionamiento. Esta última técnica es la más común y se denomina *robo de ciclo (cycle stealing)*, puesto que, en efecto, el módulo de DMA roba un ciclo de bus.

Cuando el procesador desea leer o escribir un bloque de datos, envía una orden al módulo de DMA, incluyendo la siguiente información:

- Si se solicita una lectura o una escritura, utilizando la línea de control de lectura o escritura entre el procesador y el módulo de DMA.

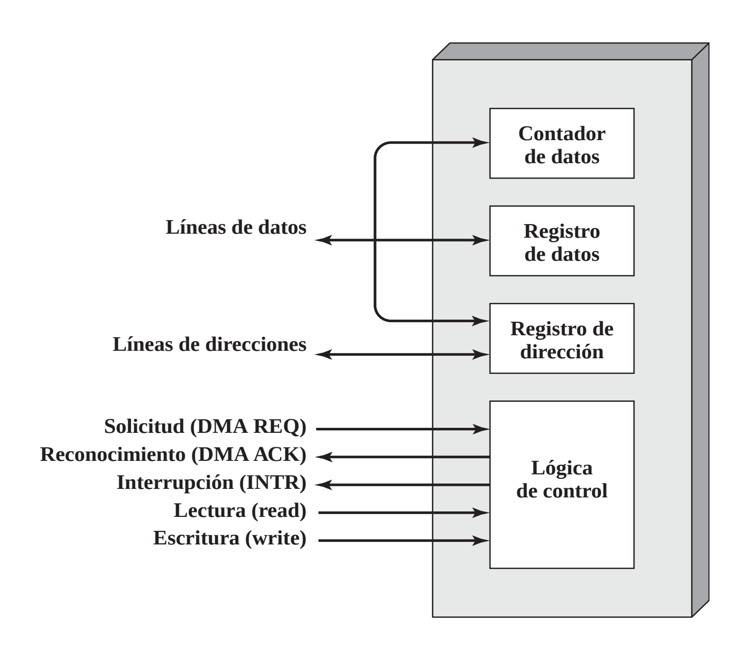


Figura 7.11. Diagrama de bloques típico de un módulo de DMA.

- La dirección del dispositivo de E/S en cuestión, indicada a través de las líneas de datos.
- La posición inicial de memoria a partir de donde se lee o se escribe, indicada a través de las líneas de datos y almacenada por el módulo de DMA en su registro de direcciones.
- El número de palabras a leer o escribir, también indicado a través de las líneas de datos y almacenado en el registro de cuenta de datos.

Después, el procesador continúa con otro trabajo. Ha delegado la operación de E/S al módulo de DMA, que se encargará de ella. El módulo de DMA transfiere el bloque completo de datos, palabra a palabra, directamente desde o hacia la memoria, sin que tenga que pasar a través del procesador. Cuando la transferencia se ha terminado, el módulo de DMA envía una señal de interrupción al procesador. Así pues, el procesador solo interviene al comienzo y al final de la tranferencia (Figura 7.5c).

La Figura 7.12 muestra en qué momento del ciclo de instrucción puede detenerse el procesador. En cada caso, el procesador se detiene justo antes de necesitar el bus. Después, el módulo de DMA transfiere una palabra y devuelve el control al procesador. Obsérvese que no se trata de una interrupción; el procesador no guarda el contexto ni hace nada más. En cambio, el procesador espera durante un ciclo de bus. El efecto resultante es que el procesador es más lento ejecutando los programas. No obstante, para una transferencia de E/S de varias palabras, el DMA es mucho más eficiente que la E/S mediante interrupciones o la E/S programada.

El mecanismo de DMA puede configurarse de diversas formas. La Figura 6.14 muestra algunas posibilidades. En el primer ejemplo, todos los módulos comparten el mismo bus del sistema. El módulo de DMA, actuando como un procesador suplementario, utiliza E/S programada para intercambiar datos entre la memoria y un módulo de E/S a través del módulo de DMA. Esta configuración, si bien es la más económica, es claramente ineficiente. Igual que con la E/S programada controlada por el procesador, la transferencia de cada palabra consume dos ciclos de bus.

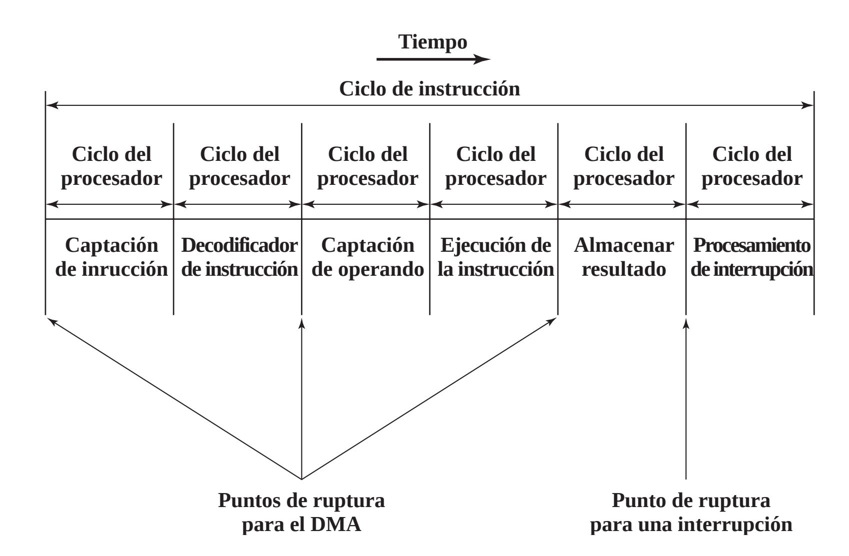


Figura 7.12. Puntos de ruptura para el DMA y las interrupciones en un ciclo de instrucción.

El número de ciclos de bus necesarios puede reducirse sustancialmente si se integran las funciones de DMA y de E/S. Como indica la Figura 7.13b, esto significa que existe un camino entre el módulo de DMA y uno o más módulos de E/S que no incluye al bus del sistema. La lógica de DMA puede ser parte de un módulo de E/S, o puede ser un módulo separado que controla a uno o más módulos de E/S. Este concepto se puede llevar algo más lejos conectando los módulos de E/S a un módulo de DMA mediante un bus de E/S (Figura 7.13c). Esto reduce a uno el número de interfaces de E/S en el módulo de DMA y permite una configuración fácilmente ampliable. En todos estos casos (Figuras 7.13b y c) el bus del sistema, que el módulo de DMA comparte con el procesador y la memoria, es usado por el módulo de DMA solo para intercambiar datos con la memoria. El intercambio de datos entre los módulos de DMA y E/S se produce fuera del bus del sistema.

CONTROLADOR DE DMA 8237A DE INTEL

El controlador de DMA 8237A de Intel proporciona la interfaz necesaria para realizar el acceso directo a la memoria DRAM en los computadores basados en procesadores de la familia 80x86. La Figura 7.14 muestra la ubicación del controlador o módulo de DMA. Cuando en módulo de DMA necesita utilizar los buses (de datos, dirección, y control) para transferir datos, envía una señal denominada HOLD (*adquisición del control del bus*) al procesador. El procesador responde con la señal HLDA (HOLD *Acknowledge*, reconocimiento de HOLD), para indicar al módulo de DMA que puede utilizar los buses. Por ejemplo, si el módulo de DMA va a transferir un bloque de datos desde la memoria al disco, realizará lo siguiente:

1. El periférico (por ejemplo un controlador de disco) solicitará el DMA activando la señal DREQ (DMA *REQuest*, solicitud de DMA).
2. El controlador de DMA activará HRQ (HOLD *ReQuest*, solicitud de HOLD) indicando a la CPU, a través de su terminal HOLD, que necesita utilizar los buses.

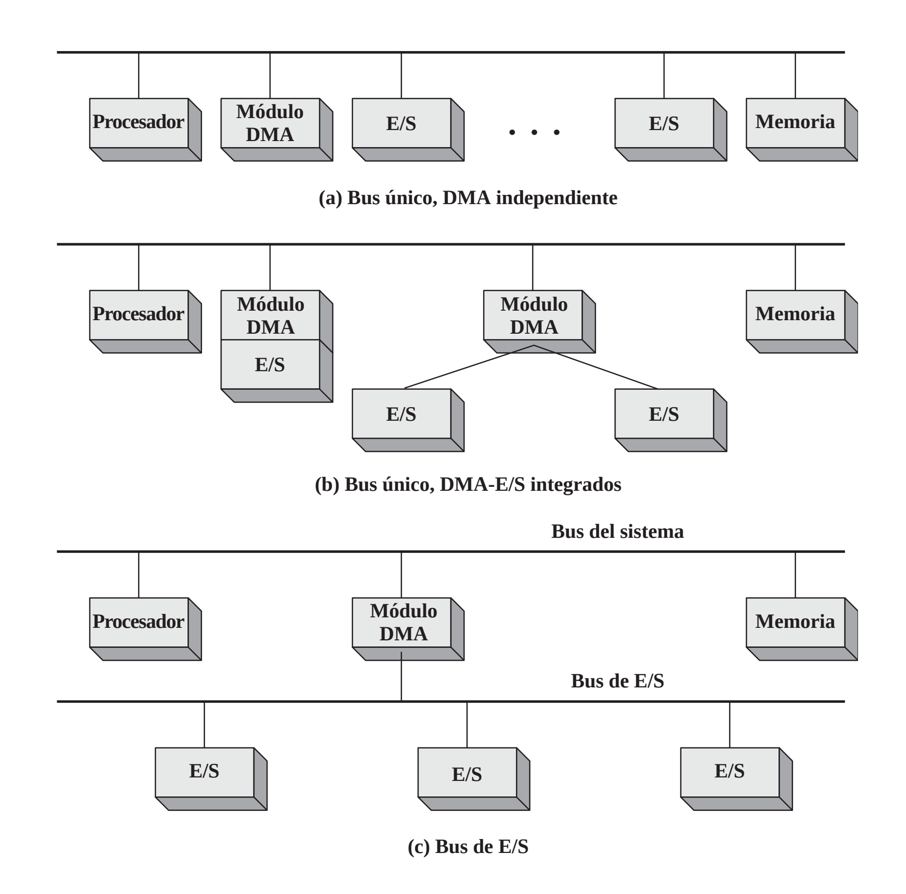


Figura 7.13. Configuraciones alternativas para el DMA.

3. La CPU terminará el ciclo de bus en el que se encuentra (no necesariamente la instrucción que se está ejecutando) y responderá a la solicitud de DMA activando HDLA (reconocimiento de HOLD) para indicar al 8237 que puede proseguir y utilizar los buses para llevar a cabo la transferencia de DMA. La señal HOLD debe mantenerse activa mientras el controlador de DMA se encuentra involucrado en la transferencia.
4. El controlador de DMA activará DACK (reconocimiento de DMA) para indicar al periférico que va a empezar a transferir datos.
5. El controlador de DMA empieza a transferir datos desde la memoria al periférico poniendo la dirección del primer byte del bloque en el bus de direcciones y activando MEMR, para así poner ese byte en el bus de datos. Después activa IOW para escribir el contenido del bus de datos en el periférico. El controlador de DMA decrementa el contador e incrementa el puntero de dirección y repite el procedimiento hasta que el contador llegue a cero y acabe la transferencia.
6. Una vez el controlador de DMA ha terminado, desactivará HRQ para indicar a la CPU que puede tomar de nuevo el control de los buses.

Mientras que el controlador de DMA está utilizando los buses para transferir los datos, el procesador se encuentra «ocioso» (*idle*). El 8237 se conoce como controlador de DMA al vuelo (*fly-by*

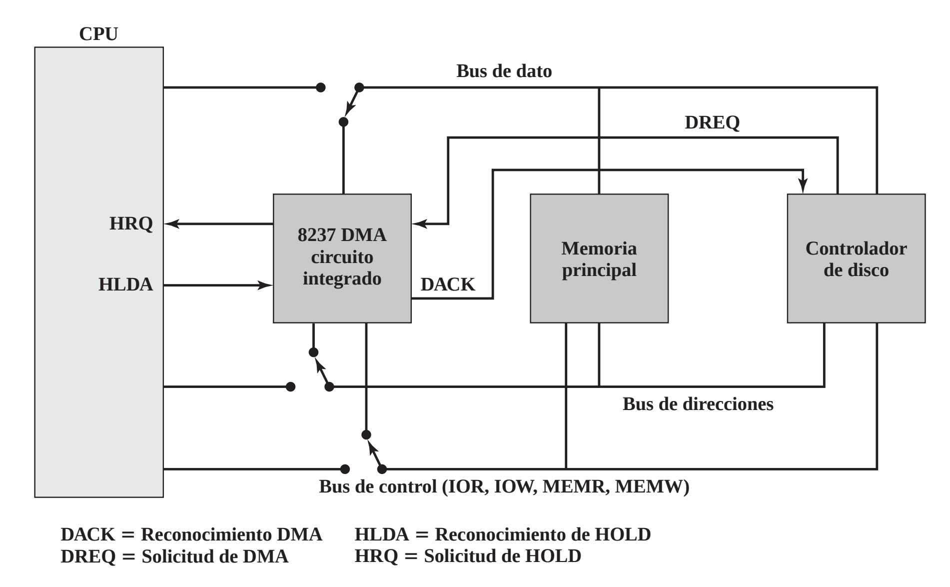


Figura 7.14. Uso del bus del sistema por el controlador de DMA 8237.

DMA) ya que los datos que se transfieren no pasan a través del propio circuito controlador de DMA y no se almacenan en el mismo. Por consiguiente, solo puede controlar transferencias entre un puerto de E/S y una dirección de memoria pero no entre dos puertos de E/S o dos posiciones de memoria. No obstante, como se explica a continuación, el controlador de DMA puede realizar una transferencia de memoria a memoria a través de un registro.

El 8237 dispone de cuatro canales de DMA, numerados como 0, 1, 2, 3, y 4, que pueden programarse independientemente. Cada uno de esos canales puede estar activo en cada momento.

El 8237 tiene un conjunto de cinco registros de control para programar y controlar la operación de DMA en cada uno de los canales (Tabla 7.4):

- **Orden (*Command*):** el procesador carga este registro para controlar la operación de DMA. D0 habilita una transferencia de memoria a memoria en la que el canal 0 se utiliza para transferir un byte en un registro temporal del 8237 y el canal 1 para transferir el byte desde el registro a memoria. Cuando la transferencia de memoria a memoria está habilitada, D1 se puede usar para deshabilitar el incremento/decremento en el canal 0 de forma que pueda escribirse un valor fijo en un bloque de memoria. D2 habilita o deshabilita el DMA.
- **Estado (*Status*):** el procesador lee este registro para determinar el estado del controlador de DMA. Los bits D0-D3 se utilizan para indicar si los canales 0-3 han alcanzado el valor final de su cuenta (TC, *Terminal Count*). El procesador utiliza los bits D4-D7 para determinar si existe una petición de DMA pendiente en alguno de los canales.
- **Modo (*Mode*):** el procesador utiliza este registro para establecer el modo de operación del controlador de DMA. Los bits D0 y D1 se utilizan para seleccionar el canal. Los otros bits establecen el modo de operación del canal seleccionado. Los bits D2 y D3 indican si la transferencia es desde un dispositivo de E/S a memoria (escritura) o desde memoria a un dispositivo de E/S (lectura), o una operación de verificación. Si D4 está activo, el registro de dirección

Tabla 7.4. Registros del 8237A.

Bit	Orden	Estado	Modo	Máscara simple	Máscara completa
D0	Habilitar/deshabilitar memoria a memoria	Canal 0 ha alcanzado el final de la cuenta	Selección de canal	Seleccionar bit de máscara de canal	Activar / desactivar bit de máscara de canal 0
D1	Habilitar/deshabilitar dirección fija para canal 0	Canal 1 ha alcanzado el final de la cuenta			Activar / desactivar bit de máscara de canal 1
D2	Habilitar/deshabilitar controlador	Canal 2 ha alcanzado el final de la cuenta	Transferencia de lectura/escritura/ verificación	Activar / desactivar Bit de máscara	Activar / desactivar bit de máscara de canal 2
D3	Temporización norma/comprimida	Canal 3 ha alcanzado el final de la cuenta		No usado	Activar / desactivar bit de máscara de canal 3
D4	Prioridad fija/rotatora	Solicitud de Canal 0	Habilitar/deshabilitar autoinicialización		No utilizado
D5	Selección de escritura extendida	Solicitud de Canal 0	Selección de incremento / decremento de dirección		
D6	Señal DREQ activa en alta/baja	Solicitud de Canal 0	Selección de modo de cascada demanda/ bloque/ un dato		
D7	Señal DACK activa en alta/baja	Solicitud de Canal 0			

de memoria y el registro contador se cargan con sus valores originales al final de la transferencia de DMA. Los bits D6 y D7 establecen la forma en que se utiliza el 8237. En el modo simple (*single mode*) se transfiere un único byte de datos. Los modos de bloque y de demanda se utilizan para transferir bloques, permitiendo el modo de demanda la finalización prematura de la transferencia. El modo en cascada permite que varios circuitos 8237 puedan ser encadenados para ampliar a más de cuatro el número de canales.

- **Máscara simple (*Single Mask*):** este registro es modificado por el procesador. Los bits D0 y D1 seleccionan el canal. El bit D2 borra o activa el bit de máscara correspondiente al canal seleccionado. A través de este registro la entrada DREQ correspondiente a un canal específico puede enmascararse (deshabilitarse). Mientras que el registro de orden puede utilizarse para deshabilitar el controlador de DMA completo, el registro de máscara simple permite que el programador habilite o deshabilite un canal específico.
- **Máscara completa (*All Mask*):** este registro es similar al registro de máscara simple, pero permite que los cuatro canales se habiliten o deshabiliten con una operación de escritura.

Además, el 8237A dispone de ocho registros de datos: un registro de dirección de memoria, y un registro contador por cada canal. El procesador utiliza estos registros para indicar la posición y el tamaño de la zona de memoria principal afectada por la transferencia.

7.6. CANALES Y PROCESADORES DE E/S

LA EVOLUCIÓN DEL FUNCIONAMIENTO DE LAS E/S

A medida que los computadores han evolucionado, la complejidad y sofisticación de sus componentes se ha incrementado. En ningún lugar se hace más evidente que en el funcionamiento de las E/S. Ya se ha considerado parte de esta evolución. Sus etapas se pueden resumir como sigue:

1. La CPU controla directamente al periférico. Esta situación se observa en los dispositivos simples controlados por microprocesadores.
2. Se añade un controlador o módulo de E/S. La CPU utiliza E/S programada sin interrupciones. De esta forma, la CPU se independiza de los detalles específicos de las interfaces de los dispositivos externos.
3. Se utiliza la misma configuración del paso 2, pero ahora se emplean interrupciones. La CPU no necesita esperar a que se realice la operación de E/S, incrementándose la eficiencia.
4. El módulo de E/S tiene acceso directo a la memoria a través del DMA. Ahora se puede transferir un bloque de datos a, o desde, la memoria sin implicar a la CPU, excepto al comienzo y al final de la transferencia.
5. El módulo de E/S se mejora haciendo que se comporte como un procesador en sí, con un repertorio especializado de instrucciones orientado a las E/S. La CPU hace que el procesador de E/S ejecute un programa de E/S en memoria. El procesador de E/S capta y ejecuta sus instrucciones sin intervención de la CPU. Esto permite que la CPU pueda especificar una secuencia de actividades de E/S y ser interrumpida cuando se haya completado la secuencia entera.
6. El módulo de E/S tiene una memoria local propia y es, de hecho, un computador en sí. Con esta arquitectura, se puede controlar un conjunto grande de dispositivos de E/S con la mínima intervención de la CPU. Un uso común de este tipo de arquitectura ha sido la comunicación con terminales interactivos. El procesador de E/S se ocupa de la mayoría de las tareas correspondientes al control de los terminales.

Siguiendo el camino marcado por esta evolución, cada vez más funciones de E/S se realizan sin la intervención de la CPU. La CPU se releva de las tareas relacionadas con las tareas de E/S, mejorando las prestaciones. Con las dos últimas etapas (5-6), se ha producido un cambio importante al introducir el concepto de un módulo de E/S capaz de ejecutar un programa. En el caso de la etapa 5, el módulo normalmente se denomina *canal de E/S*. En el paso 6, se utiliza usualmente el término *procesador de E/S*. Sin embargo, ambos términos se aplican ocasionalmente a ambas situaciones. En lo que sigue, utilizaremos el término canal de E/S.

CARACTERÍSTICAS DE LOS CANALES DE E/S

El canal de E/S representa una ampliación del concepto de DMA. Un canal de E/S puede ejecutar instrucciones de E/S, lo que le confiere un control completo sobre las operaciones de E/S. En un

computador con tales dispositivos, la CPU no ejecuta instrucciones de E/S. Dichas instrucciones se almacenan en memoria principal para ser ejecutadas por un procesador de uso específico contenido en el propio canal de E/S. De esta forma, la CPU inicia una transferencia de E/S indicando al canal de E/S que debe ejecutar un programa de la memoria. El programa especifica el dispositivo o dispositivos, el área o áreas de memoria para almacenamiento la prioridad, y las acciones a realizar en ciertas situaciones de error. El canal de E/S sigue estas instrucciones y controla la transferencia de datos.

Como ilustra la Figura 7.15, son comunes dos tipos de canales de E/S. Un *canal selector* controla varios dispositivos de velocidad elevada y, en un instante dado, se dedica a transferir datos a uno de esos dispositivos. Es decir, el canal de E/S selecciona un dispositivo y efectúa la transferencia de datos. Cada dispositivo, o pequeño grupo de dispositivos, es manejado por un *controlador*, o módulo de E/S, que es similar a los módulos de E/S que se han discutido. Así, el canal de E/S se utiliza en lugar de la CPU para controlar estos controladores de E/S. Un *canal multiplexor* puede manejar las E/S de varios dispositivos al mismo tiempo. Para dispositivos de velocidad reducida, un *multiplexor de byte* acepta o transmite caracteres tan rápido como es posible a varios dispositivos. Por ejemplo, la cadena de caracteres resultante a partir de tres dispositivos con diferentes velocidades y cadenas

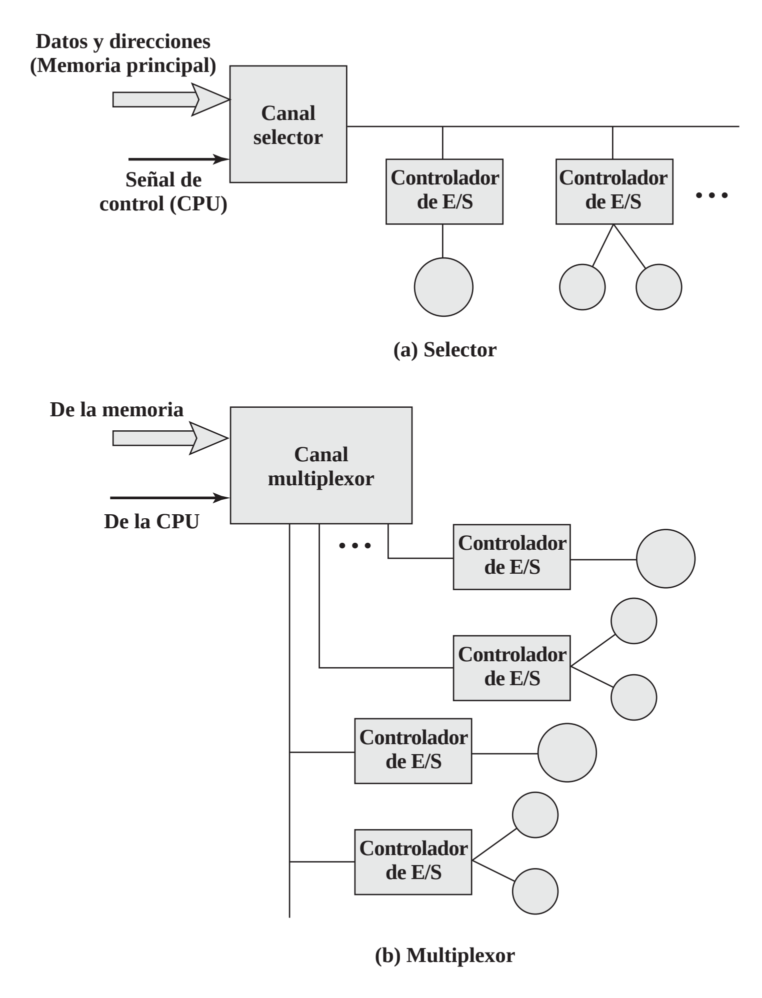


Figura 7.15. Arquitectura de un canal de E/S.

individuales $A_1A_2A_3A_4$...,$B_1B_2B_3B_4$..., y $C_1C_2C_3C_4$..., podría ser $A_1B_1C_1A_2C_2A_3B_2C_3A_4$, y así sucesivamente. Para dispositivos de velocidad elevada, un *multiplexor de bloque* entrelaza bloques de datos de los distintos dispositivos.

7.7. LA INTERFAZ EXTERNA: FIREWIRE E INFINIBAND

TIPOS DE INTERFACES

La interfaz entre el periférico y el módulo de E/S debe ajustarse a la naturaleza y la forma de funcionar del periférico. Una de las principales características de la interfaz es si es serie o paralela (Figura 7.16). En una **interfaz paralela**, hay varias líneas conectando el módulo de E/S y el periférico, y se transfieren varios bits simultáneamente a través del bus de datos. En una **interfaz serie**, hay solo una línea para transmitir los datos y los bits deben transmitirse uno a uno. Las interfaces paralelas se utilizan usualmente para los dispositivos de alta velocidad, como una cinta o un disco. Las interfaz serie son más propias de impresoras y terminales. Con la nueva generación de interfaces serie de alta velocidad, las interfaces paralelas son menos comunes actualmente.

En cualquier caso, el módulo de E/S debe establecer un diálogo con el periférico. En términos generales, el diálogo para una operación de escritura es como sigue:

1. El módulo de E/S envía una señal de control solicitando permiso para enviar datos.
2. El periférico reconoce la solicitud.
3. El módulo de E/S transfiere los datos (una palabra o un bloque según el periférico).
4. El periférico reconoce la recepción de los datos.

Una operación de lectura se realiza de forma similar.

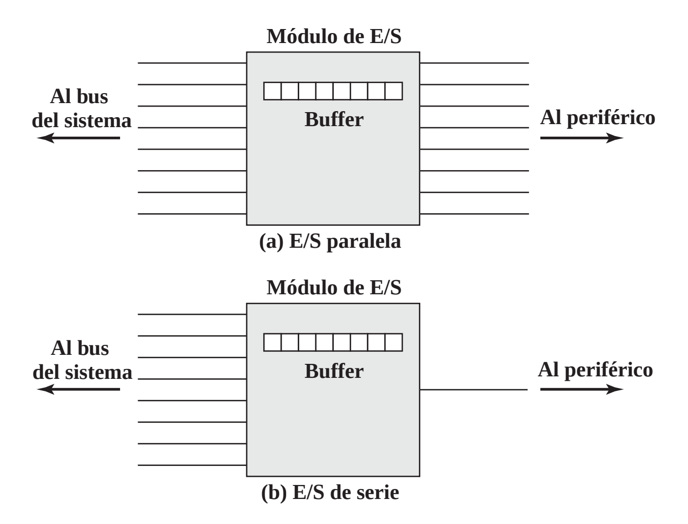


Figura 7.16. E/S paralela y serie.

Para el funcionamiento del módulo de E/S es clave disponer de un registro de acoplo (buffer) interno que pueda almacenar los datos a transferir entre el periférico y el resto del sistema. Este buffer permite que el módulo de E/S pueda compensar las diferencias de velocidad entre el bus del sistema y sus líneas externas.

CONFIGURACIONES PUNTO-A-PUNTO Y MULTIPUNTO

La conexión entre un módulo de E/S del computador y los dispositivos externos pueden ser punto-a-punto o multipunto. Una interfaz punto-a-punto proporciona una línea específica entre el módulo de E/S y el dispositivo externo. En los sistemas pequeños (PC, estaciones de trabajo), existen usualmente enlaces punto-a-punto para el teclado, la impresora, y el modem externo. Un ejemplo típico de este tipo de interfaz es la especificación EIA-232 (para su descripción, *véase* [STAL04]).

Las interfaces externas multipunto, utilizadas para soportar dispositivos de almacenamiento masivo (discos y cintas) y dispositivos multimedia (CD-ROM, equipos de vídeo y audio), tienen una importancia creciente. Estas interfaces multipunto de hecho son buses externos, y poseen el mismo tipo de lógica que los buses que se discutieron en el Capítulo 3. En esta sección, consideraremos dos ejemplos clave: FireWire e Infiniband.

BUS SERIE FIREWIRE

Con velocidades en los procesadores entorno a los GHz y dispositivos de almacenamiento con capacidades del orden de varios gigabits, las demandas de E/S de los computadores personales, las estaciones de trabajo, y los servidores son impresionantes. Las tecnologías de canales de E/S de alta velocidad que se han desarrollado para los grandes computadores centrales (*mainframes*) y los supercomputadores son todavía demasiado caras y voluminosas para que se utilicen en sistemas pequeños. En consecuencia, ha existido un gran interés en desarrollar alternativas a la SCSI y a otras interfaces de E/S para sistemas pequeños. El resultado es el estándar IEEE 1394 para un bus serie de altas prestaciones conocido como FireWire.

FireWire presenta ciertas ventajas sobre SCSI y otras interfaces de E/S. Es de muy alta velocidad, bajo costo, y fácil de implementar. De hecho, FireWire no solo se está utilizando en computadores, sino también para productos de electrónica de consumo, tales como cámaras digitales, VCR, y televisiones. En estos productos, el bus FireWire se usa para transmitir imágenes de vídeo, que provienen cada vez con más frecuencia de fuentes digitalizadas.

Una de las ventajas de la interfaz FireWire es que utiliza transmisión serie (un bit cada vez) en lugar de paralela. Las interfaces paralelas, como SCSI, necesitan más líneas, lo que significa cables más anchos y más caros, y conectores más anchos y caros con más terminales que se pueden doblar o romper. Un cable con más líneas necesita estar protegido frente a las posibles interferencias eléctricas entre las líneas. Además, con una interfaz paralela, es necesaria la sincronización entre las líneas y constituye un problema más grave a medida que aumenta la longitud del cable.

Además, los computadores son cada vez más pequeños físicamente, incluso a medida que aumentan sus prestaciones y sus necesidades de E/S. Los computadores portátiles y los de bolsillo tienen poco espacio para los conectores, pero necesitan velocidades de datos elevadas para poder manejar imágenes y vídeo.

FireWire intenta proporcionar una única interfaz de E/S con un conector sencillo que pueda manejar diversos dispositivos a través de un único puerto, de manera que los conectores del ratón, la impresora, de SCSI, del controlador de disco externo, de sonido y los de la red de área local puedan reemplazarse por este único conector. El conector se inspira en el utilizado en la videoconsola Gameboy de Nintendo. Es tan cómodo que basta con localizarlo detrás de la máquina y conectar el dispositivo sin más.

Configuraciones de FireWire. FireWire utiliza la configuración de conexión en cadena (*daisy chain*), con hasta 63 dispositivos conectados con un solo puerto. Además, pueden interconectarse mediante adaptadores (*bridges*) hasta 1022 buses FireWire, posibilitando que el sistema soporte tantos periféricos como se necesite.

FireWire soporta lo que se conoce como conexión rápida (*hot plugging*), que permite conectar y desconectar periféricos sin tener que reconfigurar el sistema. Además, FireWire permite la configuración automática; y no es necesario fijar manualmente los identificadores (ID) del dispositivo o tener en cuenta la posición relativa de los dispositivos. La Figura 7.17 muestra una configuración FireWire sencilla. En el bus FireWire no hay terminadores, y el sistema realiza automáticamente la configuración para asignar direcciones. Obsérvese, además, que los dispositivos en el bus FireWire no necesitan conectarse estrictamente en cadena (*daisy chain*). Se trata más bien de una configuración con estructura de árbol.

Una propiedad importante del estándar FireWire es que especifica un conjunto de protocolos de tres capas para estandarizar la forma en la que el computador anfitrión interactúa con los dispositivos periféricos a través del bus serie. La Figura 7.18 ilustra estas capas. Las tres capas son:

- **Capa física:** define los medios de transmisión que se permiten para FireWire y las señales y características eléctricas de cada uno de ellos.
- **Capa de enlace:** describe la transmisión de los datos en los paquetes.
- **Capa de transacción:** define un protocolo de solicitud-respuesta que oculta a las aplicaciones los detalles de las capas inferiores.

Capa física. La capa física del bus FireWire especifica varios medios de transmisión alternativos y sus conectores, con diferentes propiedades físicas y de transmisión de datos. Se definen velocidades

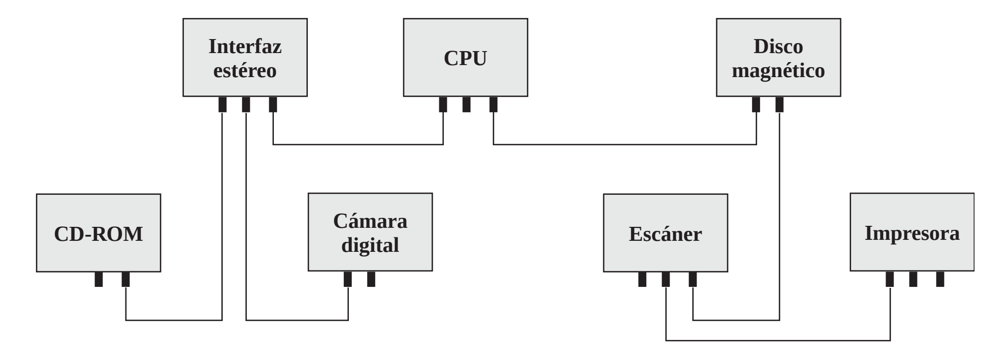


Figura 7.17. Configuración FireWire sencilla.

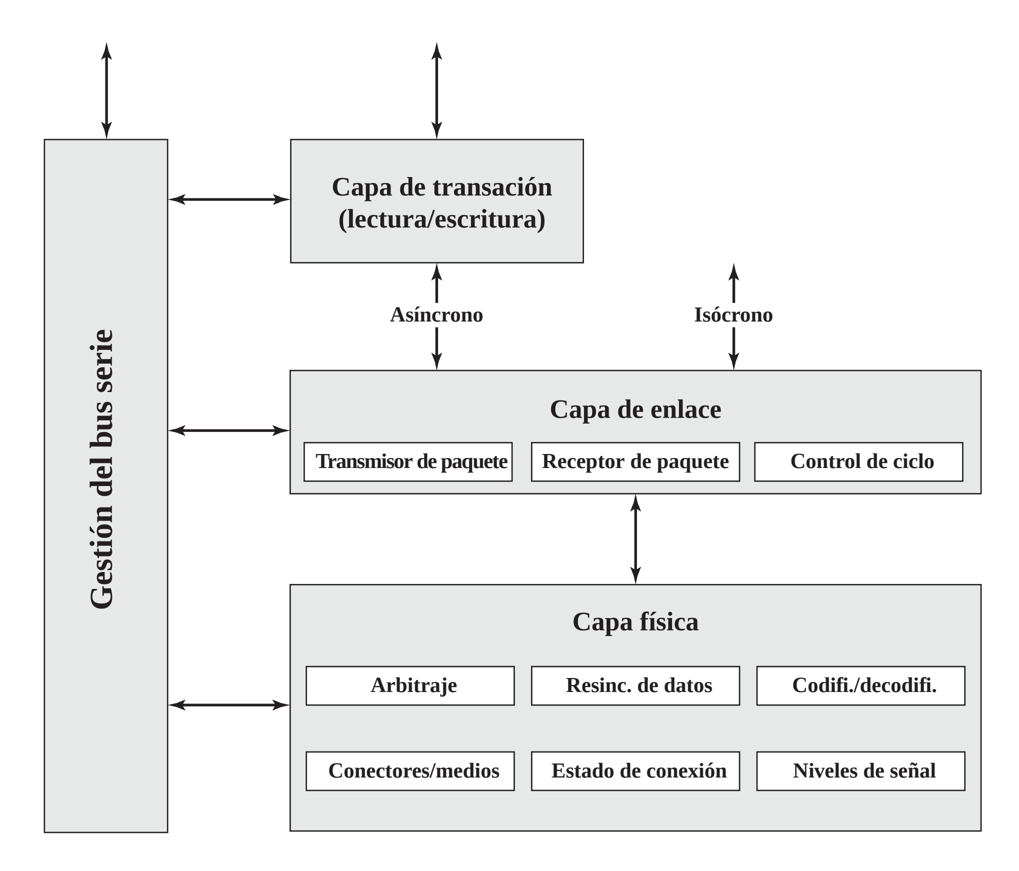


Figura 7.18. Pila del protocolo FireWire.

de datos desde 25 a 400 Mbps. La Capa Física transforma los datos binarios en las señales eléctricas precisas para diversos medios físicos. Esta capa también proporciona el servicio de arbitraje que garantiza que sólo un dispositivo transmitirá datos en cada momento.

FireWire proporciona dos formas de arbitraje. La forma más simple se basa en la estructura de tipo árbol de los nodos del bus FireWire, mencionada antes. Un caso especial de esta estructura es la conexión lineal en cadena (*daisy chain*). La capa física incluye la lógica que permite que todos los dispositivos conectados se configuren de manera que un nodo sea designado como la raiz del árbol y los otros se organicen mediante relaciones de padre/hijo conformando la topología de tipo árbol. Una vez que se ha establecido la configuración, el nodo raíz actúa como un árbitro central y procesa las solicitudes de acceso al bus de manera que la primera en llegar es la primera en atenderse (*first-come-first-served*). En el caso de solicitudes simultáneas, se cede el acceso al nodo con mayor prioridad natural. La prioridad natural es mayor cuanto mayor sea la cercanía al nodo raíz, y entre los nodos que están a igual distancia, es mayor para aquel que tenga el menor valor del número de identificación (ID).

El método anterior de arbitraje se complementa con dos funciones adicionales: arbitraje equitativo y arbitraje urgente. Con el arbitraje equitativo, el tiempo del bus se organiza en *intervalos de equidad*. Al comienzo de un intervalo, cada nodo activa un bit de autorización de arbitraje. Durante el intervalo, cada uno de los nodos puede competir por el acceso al bus.

Cuando un nodo gana el acceso al bus, desactiva su bit de autorización de arbitraje y no puede competir de nuevo por el acceso al bus durante el intervalo actual. Este esquema hace que el arbitraje sea más equitativo, puesto que evita que uno o más dispositivos de prioridad alta puedan monopolizar el uso del bus.

Junto con este esquema de igualdad, algunos dispositivos se pueden configurar para que tengan una prioridad *urgente.* Estos nodos pueden ganar el acceso al bus varias veces durante un intervalo de equidad. En esencia, se trata de utilizar un contador en cada nodo de prioridad elevada que autoriza a estos nodos a controlar el 75 por ciento del tiempo del bus. Por cada paquete que se transmite como no urgente se pueden transmitir tres paquetes urgentes.

Capa de enlace. La capa de enlace define la transmisión de los datos en forma de paquetes. Se permiten dos tipos de transmisión:

- **Asíncrona:** se transmite un paquete constituido por una cantidad variable de datos y varios bytes de la capa de transacción a una dirección explícita, y se devuelve información de reconocimiento.
- **Isócrona:** se transmite una cantidad variable de datos mediante una secuencia de paquetes de tamaño fijo, a intervalos regulares. Esta forma de transmisión usa un direccionamiento simplificado y no utiliza reconocimiento.

La transmisión asíncrona se utiliza para datos que no necesitan tener una velocidad de transferencia fija. Tanto el esquema de arbitraje equitativo como el urgente pueden utilizarse para la transmisión asíncrona. El método implícito es el arbitraje equitativo. Los dispositivos que necesitan una fracción sustancial de la capacidad del bus o presentan requisitos de retardo exigentes utilizan el método de arbitraje urgente. Por ejemplo, un nodo que debe absorber datos en tiempo real a elevadas velocidades puede utilizar arbitraje urgente cuando los buffers de datos están a la mitad de su capacidad máxima.

La Figura 7.19a describe una transacción asíncrona típica. El proceso de enviar un paquete se llama subacción (*subaction*). La subacción consta de cinco periodos de tiempo:

- **Secuencia de arbitraje:** es el intercambio de señales que se precisa para ceder el control del bus a un dispositivo.
- **Transmisión de paquete:** todo paquete incluye una cabecera conteniendo los identificadores (ID) de la fuente y el destino. La cabecera también contiene información acerca del tipo de paquete, un código de redundancia cíclica (CRC), e información de los parámetros del tipo específico de paquete. Un paquete puede incluir también un bloque de datos formado por datos de usuario y otro código CRC.
- **Intervalo de reconocimiento:** este es el retardo necesario para que en el destino se reciba y decodifique un paquete, y se genere el reconocimiento.
- **Reconocimiento:** el receptor del paquete devuelve un paquete de reconocimiento con un código indicando la acción realizada por el receptor.
- **Intervalo de subacción:** es un periodo forzoso de inactividad para asegurar que ningún nodo del bus empieza el arbitraje antes de que el paquete de reconocimiento haya terminado de transmitirse.

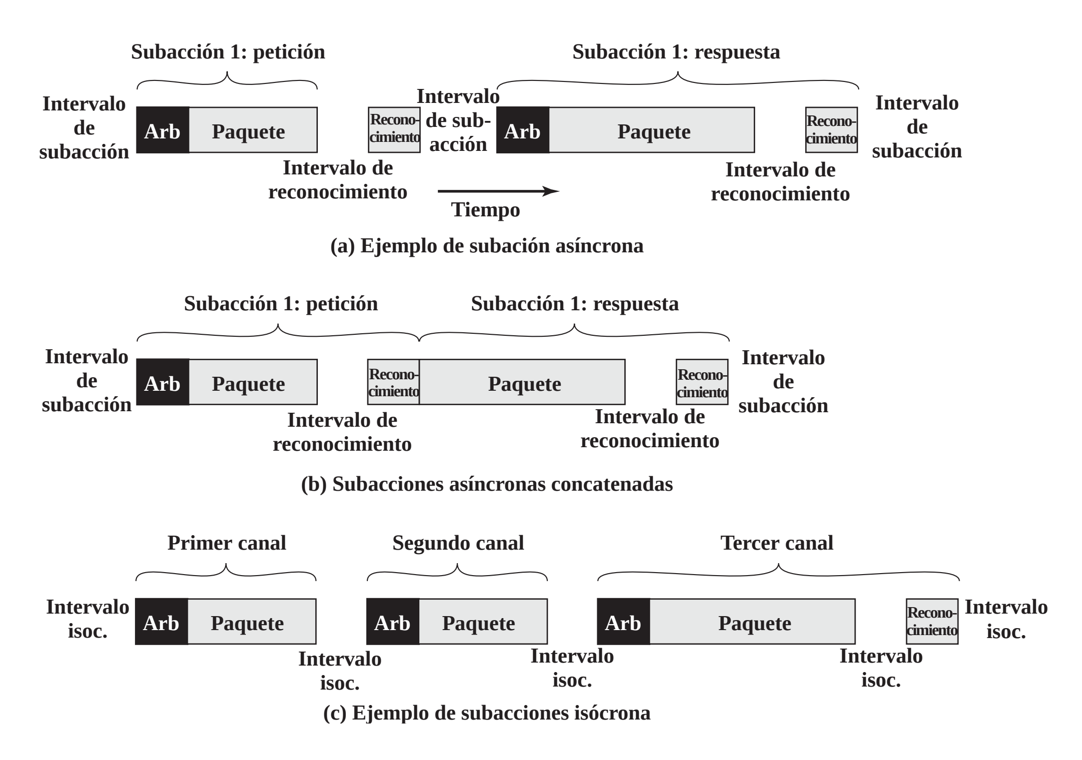


Figura 7.19. Subacciones FireWire.

En el momento en que se envía el reconocimiento, el nodo que lo está enviando tiene el control del bus. Puesto que el intercambio es una interacción de petición/respuesta entre dos nodos, el nodo que responde puede transmitir inmediatamente el paquete de respuesta sin tener que realizar una secuencia de arbitraje (Figura 7.19b).

Existe la posibilidad de acceso isócrono para los dispositivos que generan o consumen datos de manera regular, tales como los de sonido o vídeo digital. Este método asegura que los datos pueden generarse dentro de unos intervalos especificados para garantizar cierta velocidad de datos.

Para permitir una carga mixta de tráfico de datos isócronos y asíncronos, uno de los nodos se designa como *maestro de ciclo*. Periódicamente, el maestro de ciclo genera un paquete de comienzo de ciclo. Este indica a los otros nodos que se ha iniciado un ciclo isócrono. Durante este ciclo, sólo se pueden enviar paquetes isócronos (Figura 7.19c). Cada fuente de datos isócronos interviene en el arbitraje para acceder al bus. El nodo ganador transmite un paquete inmediatamente. No existe reconocimiento para este paquete, y de esta forma otras fuentes de datos isócronas arbitran el acceso al bus inmediatamente después de que se haya transferido el paquete previo. Como resultado existe un pequeño intervalo, determinado por los retardos del bus, entre la transmisión de un paquete y el periodo de arbitraje para el siguiente paquete. Este retardo, denominado intervalo isócrono, es menor que el intervalo de subacción.

Después de que todas las fuentes isócronas hayan transmitido, el bus permanecerá inactivo el tiempo suficiente para que se produzca un intervalo de subacción. Esta es la señal para que las fuentes asíncronas puedan competir por acceder al bus. Las fuentes asíncronas pueden utilizar el bus entonces hasta el comienzo del siguiente ciclo isócrono.

Los paquetes isócronos se etiquetan con números de canal de ocho bits que se asignan previamente mediante un diálogo entre los dos nodos que intercambian datos isócronos. La cabecera, que es más corta que la de los paquetes asíncronos, también incluye un campo de longitud de datos y un CRC para la cabecera.

INFINIBAND

InfiniBand es una especificación de E/S reciente orientada al mercado de servidores de gama alta[4]. La primera versión de la especificación apareció a comienzos de 2001 y atrajo a numerosos fabricantes. El estándar describe la arquitectura y las especificaciones para el flujo de datos entre procesadores y dispositivos de E/S inteligentes. Se pretendía que InfiniBand sustituyera al bus PCI en servidores, al proporcionar más capacidad, mayores posibilidades de expansión, y un aumento de la flexibilidad de diseño de los equipos. Esencialmente, InfiniBand permite que los servidores, los equipos de almacenamiento remoto y otros dispositivos de red se puedan conectar a través de un sistema de conmutadores (*switch fabric*) y enlaces. Esta arquitectura basada en conmutadores puede conectar hasta 64 000 servidores, sistemas de almacenamiento y dispositivos de red.

La arquitectura de InfiniBand. Aunque PCI constituye un procedimiento de interconexión fiable y continúa proporcionando velocidades cada vez mayores, hasta 1 Gbps, presenta una arquitectura limitada si se compara con InfiniBand. Con InfiniBand no es necesario que el hardware de interfaz de E/S se encuentre dentro del chasis del servidor. Además, en InfiniBand el acceso a almacenamiento remoto, y las conexiones a red y entre los servidores se llevan a cabo conectando todos los equipos a un sistema de conmutadores (*switch fabric*) y enlaces. Al sacar las E/S del chasis de los servidores, estos pueden hacerse más compactos. Con ello se incrementa la flexibilidad y escalabilidad de los centros de procesamiento de datos, que pueden incorporar nuevos equipos a medida que sea necesario.

A diferencia de PCI, en cuyo ámbito se miden las distancias a la placa base en centímetros, el diseño de canal de InfiniBand permite que los dispositivos de E/S puedan situarse a una distancia del servidor de hasta 17 m con cobre, de hasta 300 m con fibra óptica multimodal, y de hasta diez km con fibra óptica unimodal. Pueden alcanzarse velocidades de transmisión elevadas, de hasta 30 Gbps.

La Figura 7.20 muestra la arquitectura de InfiniBand . Los elementos clave son los siguientes:

- **Adaptador de canal de computador (*Host Channel Adapter,* HCA):** en lugar de un conjunto de ranuras PCI, un servidor típico necesita una interfaz a un HCA que conecta el servidor a un conmutador InfiniBand. El HCA se conecta al servidor a través de un controlador de memoria que tiene acceso al bus del sistema y controla el tráfico entre el procesador y la memoria, y entre el HCA y la memoria. El HCA utiliza acceso directo a memoria DMA para leer y escribir en memoria.
- **Adaptador de canal de dispositivo (*Target Channel Adapter,* TCA):** se utiliza para conectar los sistemas de almacenamiento, los encaminadores y otros dispositivos periféricos a un conmutador InfiniBand.

---

[4] InfiniBand es el resultado de la fusión de dos proyectos competidores: Future I/O (promovido por CISCO, HP, Compaq, e IBM) y Next Generation I/O (desarrollado por Intel y promovido por algunas compañías más).

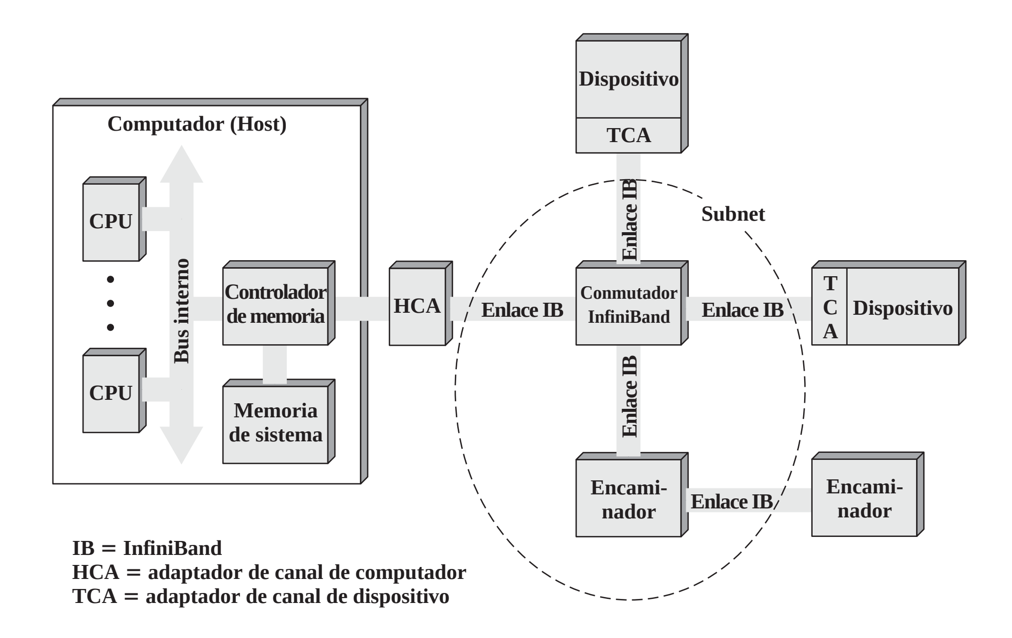


Figura 7.20. Conmutador Infiniband.

- **Conmutador InfiniBand:** conmutador que proporciona conexiones físicas punto-a-punto a un conjunto diverso de dispositivos y conmuta el tráfico entre enlaces. Los servidores y los dispositivos se comunican vía el conmutador a través de sus respectivos adaptadores. La lógica del conmutador gestiona las conexiones sin interrumpir a los servidores.
- **Enlaces:** un enlace conecta un conmutador y un adaptador de canal, o dos conmutadores.
- **Subred:** una subred consiste en uno o más conmutadores interconectados más los enlaces que conectan otros dispositivos a dichos conmutadores. La Figura 7.20 muestra una subred con un único conmutador, aunque se necesitan subredes más complejas en el caso de que haya que interconectar un número de dispositivos elevado. Las subredes permiten que los administradores de los sistemas puedan confinar las transmisiones de uno a muchos (*multicast*) y de uno a todos (*broadcast*) dentro de la subred.
- **Encaminador:** conecta las subredes InfiniBand, o conecta un conmutador InfiniBand a una red de área local, de área amplia o de dispositivos de almacenamiento.

Los adaptadores de canal son dispositivos inteligentes que gestionan todas las funciones de E/S sin la necesidad de interrumpir al procesador del servidor. Por ejemplo, hay un protocolo de control mediante el cual un conmutador detecta todos los TCA y HCA conectados a un conmutador y asigna direcciones lógicas a cada uno. Esto se lleva a cabo sin la intervención del procesador.

El conmutador de InfiniBand abre temporalmente los canales entre el procesador y los dispositivos con los que se está comunicando. Los dispositivos no tienen que compartir la capacidad de canal, como en el caso de utilizar un bus como PCI, que necesita un procedimiento de arbitraje para que los dispositivos accedan al procesador. Para añadir más dispositivos a la configuración se conecta el TCA de cada uno de ellos al conmutador.

Funcionamiento de InfiniBand. Cada enlace físico entre un conmutador y una interfaz (HCA o TCA) conectada a él puede incluir hasta 16 canales lógicos, denominados **líneas virtuales** (*virtual lanes*). Una línea se reserva para la gestión del conmutador y las otras para el transporte de los datos. Los datos se envían en forma de secuencia de paquetes, conteniendo cada paquete una parte del volumen total de datos a transferir más la información de direccionamiento y control. La transferencia de datos se gestiona mediante los correspondientes protocolos de comunicación. Una línea virtual se dedica temporalmente a transferir los datos desde un nodo a otro a través del conmutador InfiniBand. El conmutador InfiniBand asigna el tráfico que llega a través de una línea a alguna de las líneas de salida según el camino que deben seguir los datos entre los nodos que se comunican.

La Figura 7.21 describe la estructura lógica utilizada para hacer posible la comunicación a través de InfiniBand. Para dar cabida al hecho de que algunos dispositivos pueden enviar datos más rápidamente que pueden ser recibidos en el destino correspondiente, en los extremos de cada enlace existe una pareja de colas que almacenan el exceso de datos de entrada y salida. Las colas pueden ubicarse en el adaptador de canal o en la memoria del dispositivo conectado al mismo. Cada línea virtual utiliza una pareja de colas diferente. Los computadores utilizan estas colas de la siguiente forma. El computador envía una transacción, denominada elemento de la cola de trabajos (*Work Queue Entry*, WQE) a la cola de envío o recepción de su pareja de colas. Los dos WQE más importantes son las transacciones SEND (enviar) y el RECEIVE (recibir). En el caso de una operación SEND, el WQE especifica un bloque de datos en el espacio de memoria que es el que hay que enviar al destino. Un WQE correspondiente a un RECEIVE especifica el hardware donde hay que ubicar los datos que se reciban de otro dispositivo cuando este ejecute la correspondiente transición SEND. El adaptador de canal procesa cada WQE enviado según el orden de prioridad establecido y genera un elemento en la cola de trabajos concluidos (*Completion Queue Entry*, CQE) para indicar el estado de trabajo finalizado.

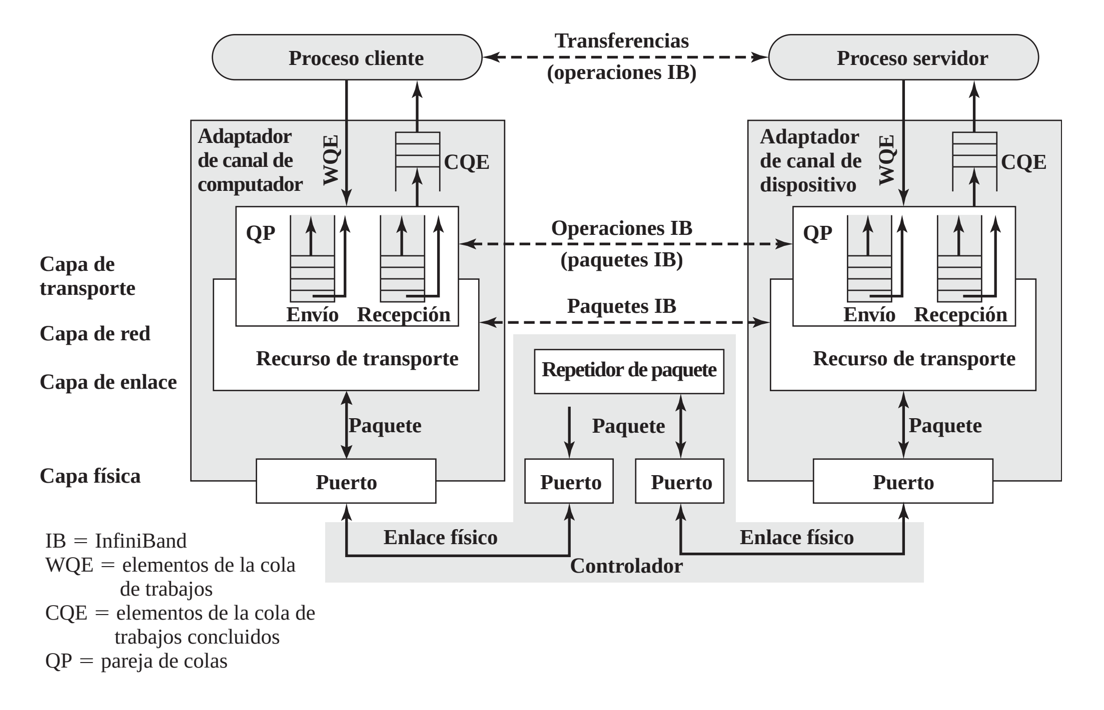


Figura 7.21. Capas del protocolo de comunicación Infiniband.

Tabla 7.5. Enlaces y anchos de banda en InfiniBand.

Enlace (ancho)	Ancho de banda de señal (unidireccional)	Capacidad utilizable (80% del ancho de banda de la señal)	Ancho de banda efectivo (envío + recepción)
1 bit	2.5 Gbps	2 Gbps (250 MBps)	(250 + 250) MBps
4 bits	10 Gbps	8 Gbps (1 GBps)	(1 + 1) GBps
12 bits	30 Gbps	24 Gbps (3 GBps)	(3 + 3) GBps

La Figura 7.21 también pone de manifiesto que se utiliza arquitectura de protocolo con cuatro capas:

- **Física:** la especificación de la capa física define tres velocidades de conexión (1X, 4X, y 12X) correspondientes a 2.5, 10, y 30 Gbps, respectivamente (Tabla 7.5). La capa física también define los medios físicos, que incluyen la fibra óptica y el cobre.
- **Enlace:** esta capa define la estructura básica de los paquetes utilizados para intercambiar datos, incluyendo el esquema de dirección que asigna una única dirección de enlace a cada dispositivo de la subred. Este nivel incluye la lógica para configurar las líneas virtuales a para la conmutación de los datos entre origen y destino a través de los conmutadores de una subred. La estructura del paquete incluye un código de detección de error que proporciona fiabilidad a este nivel.
- **Red:** la capa de red encamina los paquetes entre subredes InfiniBand diferentes.
- **Transporte:** la capa de transporte proporciona un mecanismo de fiabilidad para las transferencias punto-a-punto a través de una o más subredes.

7.8. LECTURAS Y SITIOS WEB RECOMENDADOS

Una buena descripción de los módulos de E/S y la arquitectura de Intel, incluyendo el 82C59A, el 82C55A, y el 8237A puede encontrarse en [MAZI03].

FireWire se trata con gran detalle en [ANDE98], [WICK97] y [THOM00] proporcionan una concisa descripción de FireWire.

Infiniband se describe con detalle en [SHAN03] y [FUTR01]. En [KAGA01] se puede encontrar una descripción resumida del mismo.

ANDE98 ANDERSON, D.: *FireWire System Architecture.* Reading, MA. Addison-Wesley, 1998.

FUTR01 FUTRAL, W.: *InfiniBand Architectute: Development and Deployment.* Hillsboro, OR. Intel Press, 2001.

KAGA01 KAGAN, M.: «InfinitiBand: Tkinking Outside the Box Desing». *Communications System Desing,* septiembre, 2001. **(www.csdmag.com).**

MAZI03 Mazidi, M. y Mazidi, J.: *The 80x86 IBM PC and Compatible Computers: Assembly Language, Desing and Interfacing.* Upper Saddle River, NJ. Prentice Hall, 2003.

SHAN03 Shanley, T.: *InfiniBand Network Architecture.* Teading, MA. Addison-Wesley, 2003.

THOM00 Thompson, D.: «IEEE 1394: Changing the Way Do Multimedia Communications». *IEEE Multimedia,* abril-junio, 2000.

WICK97 Wickelgren, I.: «The Facts About Fire Wire». *IEEE Spectrum,* abril, 1997.



SITIOS WEB RECOMENDADOS

- **T10 Home Page:** T10 es un comité técnico del Comité Nacional de Estándares en Tecnología de la Información (National Committee on Information Technology Standards) responsable de las interfaces de bajo nivel. Su principal trabajo es la interfaz SCSI.
- **1394 Trade Association:** incluye información técnica y punteros a proveedores de FireWire.
- **Infiniband Trade Association:** incluye información técnica y direcciones de suministradores de Infiniband
- **Caracterización y Optimización de E/S (I/O Characterization and Optimization):** página dedicada a la educación e investigación en el diseño y en las prestaciones de E/S. Herramientas y documentos útiles. Gestionada por la Universidad de Illinois.

7.9. PALABRAS CLAVE, CUESTIONES Y PROBLEMAS

PALABRAS CLAVE

Acceso Directo a Memoria (DMA)	E/S asignada en memoria	Infiniband
canal de E/S	E/S paralela	interrupción
canal multiplexor	E/S por interrupciones	módulo de E/S
canal selector	E/S programada	orden de E/S
dispositivo periférico	E/S serie	procesador de E/S
E/S aislada	FireWire	robo de ciclo

CUESTIONES

7.1. Enumere tres clasificaciones generales de dispositivos externos o periféricos.

7.2. ¿Qué es el IRA (*International Reference Alphabet*)?

7.3. ¿Cuáles son las principales funciones de un módulo de E/S?

7.4. Enumere y defina brevemente tres técnicas para realizar la E/S.

7.5. ¿Cuál es la diferencia entre E/S asignada en memoria y E/S aislada?

7.6. Cuando se produce una interrupción de dispositivo, ¿cómo determina el procesador el dispositivo que la ha originado?

7.7. Cuando un módulo de DMA toma el control del bus, y mientras mantiene dicho control, ¿qué hace el procesador?

PROBLEMAS

7.1. En un microprocesador típico, se utiliza una dirección de E/S para hacer referencia a los registros de datos de E/S y otra dirección distinta para los registros de estado y control del controlador de E/S del dispositivo correspondiente. Esos registros se denominan **puertos**. En el 8088 de Intel se utilizan dos formatos de instrucción. En un formato, un código de operación de ocho bits especifica la operación de E/S, seguido de ocho bits para la dirección del puerto. En los otros códigos de operación de E/S, la dirección del puerto se encuentra en el registro de 16 bits DX. ¿Cuántos puertos puede direccionar el 8088 en cada uno de los modos de direccionamiento de E/S?

7.2. En la familia de microprocesadores Z8000 de Zilog se utiliza un formato de instrucción similar. En este caso, es posible realizar un direccionamiento directo de los puertos utilizando una dirección de 16 bits que forma parte de la instrucción, y también un direccionamiento indirecto, al incorporar la instrucción una referencia a uno de los 16 registros de propósito general que contiene la dirección del puerto. ¿Cuántos puertos puede designar el Z8000 en cada uno de los modos de direccionamiento de E/S?

7.3. El Z8000 también incluye la posibilidad de realizar transferencias de bloques de E/S bajo el control directo del procesador (en vez de ser realizadas por DMA). Las instrucciones de transferencia de bloque especifican un registro de dirección de puerto (Rp), un registro contador (Rc) y un registro de destino (Rd). Rd contiene la dirección de memoria principal en la que debe almacenarse el primer byte leído del puerto de entrada. Rc es uno de los 16 registros de propósito general. ¿Qué tamaño de bloque de datos puede transferirse?

7.4. Considere que un microprocesador tiene una instrucción de transferencia de bloques E/S similar a la del Z8000. Tras su primera ejecución, la mencionada instrucción necesita cinco ciclos de reloj para volverse a ejecutar. En cambio, una instrucción de E/S que no transfiera bloques tardaría un total de veinte ciclos en captarse y ejecutarse. Calcule el incremento de velocidad que se obtiene con la instrucción de transferencia de bloques de E/S para un bloque de 128 bytes.

7.5. Un sistema está basado en un microprocesador de ocho bits y tiene dos dispositivos de E/S. Los controladores de E/S de este sistema utilizan registros de estado y control separados. Ambos dispositivos transfieren los datos byte a byte. El primer dispositivo tiene dos líneas de estado y tres de control. El segundo tres líneas de estado y cuatro de control.

(a) ¿Cuántos registros de ocho bits de control de módulo de E/S se necesitan para leer el estado de cada dispositivo y controlarlo?

(b) ¿Cuántos registros de control de módulo se necesitan si se supone que el primer dispositivo es un dispositivo de salida?

(c) ¿Cuántas direcciones distintas hacen falta para controlar los dos dispositivos?

7.6. En el caso de la E/S programada, la Figura 7.5 pone de manifiesto que el procesador se encuentra en un bucle de espera comprobando el estado del dispositivo de E/S. Para incrementar la eficiencia, se podría hacer el software de E/S de forma que se comprobase el estado del dispositivo periódicamente. Si el dispositivo no está preparado, el procesador podría pasar a realizar otras tareas. Después de un cierto intervalo, el procesador volvería a comprobar el estado nuevamente.

(a) Considere el esquema anterior para enviar un carácter en cada operación de salida a una impresora de diez caracteres por segundo (cps). ¿Qué pasaría si su estado se comprobase cada 200 ms?

(b) A continuación, considere un teclado con un buffer para un solo carácter. En promedio, los caracteres se introducen a una velocidad de 10 cps. Sin embargo, el tiempo entre dos pulsaciones de tecla consecutivas puede ser de solo 60 ms. ¿Con qué frecuencia debería el programa de E/S comprobar el teclado?

7.7. Un microprocesador comprueba el estado de un dispositivo de salida cada 20 ms. Esto se lleva a cabo mediante un temporizador que avisa al procesador cada 20 ms. La interfaz del dispositivo tiene dos puertos: uno para el estado y otro para el dato de salida. ¿Cuánto se tarda en comprobr el estado y atender al dispositivo si la frecuencia de reloj es de 8 MHz? Por simplicidad, considere que todas las instrucciones que intervienen tardan doce ciclos de reloj.

7.8. En la Sección 7.3 se enumeraron una ventaja y un inconveniente de la E/S asignada en memoria, comparada con la E/S aislada. Enumere dos ventajas y dos inconvenientes más.

7.9. Un sistema es controlado por un operador a través de una serie de comandos que se introducen desde un teclado. En cada intervalo de ocho horas se introducen un promedio de sesenta comandos.

(a) Suponga que el procesador comprueba el teclado cada 100 ms. ¿Cuántas veces se chequea en un periodo de ocho horas?

(b) ¿En qué porcentaje se reduciría el número de comprobaciones de teclado si se utilizase E/S por interrupciones?

7.10. Considere un sistema que utiliza E/S por interrupciones para un dispositivo que transfiere continuamente un promedio de 8 KB/s.

(a) Suponga que el procesamiento de la interrupción necesita alrededor de cien microsegundos (es decir, el tiempo se saltar a la rutina de servicio, ISR, ejecutarla y regresar al programa principal). Determine qué fracción del tiempo del procesador se dedica al dispositivo de E/S si este genera una interrupción por cada byte.

(b) Considere ahora que el dispositivo tiene dos bufferes de 16 bits e interrumpe al procesador cuando uno de ellos está lleno. Por supuesto, el procesamiento de la interrupción tarda más puesto que deben transferirse 16 bytes. Al ejecutar la ISR, el procesador tarda unos ocho microsegundos para transferir cada bytes. Determine qué fracción del tiempo del procesador se dedica al dispositivo de E/S en este caso.

(c) Ahora suponga que el procesador dispone de instrucciones de transferencia de bloques como las que tiene el Z8000. Esto permite que la ISR pueda transferir cada byte en solo dos microsegundos. Determine la fracción del tiempo del procesador que se dedica al dispositivo de E/S en este caso.

7.11. En casi todos los sistemas que tienen módulos de DMA, el acceso del módulo de DMA a memoria principal tiene más prioridad que el acceso de la CPU a memoria principal. ¿Por qué?

7.12. Un módulo de DMA transfiere caracteres a memoria mediante robo de ciclo desde un dispositivo que transmite a 9600 bps. El procesador capta instrucciones a un ritmo de un millón por segundo (1 MIPS). ¿Cuánto disminuye la velocidad del procesador debido al DMA?

7.13. Considere un sistema en el que una transferencia a través del bus necesita 500 ns. La transferencia de control del bus en uno u otro sentido, entre el procesador y un dispositivo de E/S, necesita 250 ns. Uno de los dispositivos de E/S tiene una velocidad de transferencia de 50 KB/s y utiliza DMA. Los datos se transfieren byte a byte.

(a) Suponga que empleamos DMA en modo de ráfaga. Es decir, la interfaz de DMA adquiere el control del bus antes de empezar la transferencia de un bloque y mantiene dicho control durante la transferencia completa. ¿Durante cuánto tiempo el dispositivo tiene el bus ocupado si se transfieren 128 bytes?

(b) Repita el cálculo si se utiliza el modo de robo de ciclo.

7.14. A partir del diagrama de tiempos del 8237A se observa que, una vez que comienza la transferencia de un bloque, se necesitan tres ciclos de reloj de bus por ciclo de DMA. Durante el ciclo de DMA, el 8237A transfiere un byte de información entre la memoria y el dispositivo de E/S.

(a) Suponga que el reloj del 8237A tiene una frecuencia de 5 MHz. ¿Cuánto tiempo se necesita para transferir un byte?

(b) ¿Cuál es la velocidad de transferencia máxima que se puede conseguir?

(c) Considere que la memoria no es lo suficientemente rápida y que hay que insertar dos estados de espera por ciclo de DMA. ¿Cuál será la velocidad de transferencia en este caso?

7.15. Suponga que en el sistema del problema anterior, el ciclo de memoria es de 750 ns. ¿Cuánto es posible reducir la frecuencia de reloj de bus sin que tenga efecto en la velocidad de transferencia de datos que se puede alcanzar?

7.16. Un controlador de DMA atiende cuatro enlaces de comunicación de entrada (uno por canal de DMA) con una velocidad de 64 Kbps cada uno.

(a) ¿Qué modo seleccionaría para el controlador, robo de ciclo o ráfaga?

(b) ¿Qué esquema de prioridad utilizaría para atender a los canales de DMA?

7.17. Un computador de 32 bits tiene dos canales selectores y un canal multiplexor. Cada canal selector soporta dos discos magnéticos y dos unidades de cinta magnética. El canal multiplexor tiene conectadas dos impresoras de línea, dos lectoras de tarjetas y diez terminales VDT. Suponga las siguientes velocidades de transferencia:

Unidad de Disco	800 KBytes/s
Unidad de Cinta Magnética	200 KBytes/s
Inpresora de Línea	6.6 KBytes/s
Lector de Tarjetas	1.2 KBytes/s
VDT	1 KBytes/s

Estime la máxima velocidad total de transferencia de E/S en el sistema.

7.18. Un computador está constituido por un procesador y un dispositivo D de E/S conectado a la memoria principal M a través de un bus compartido de una palabra. El procesador puede ejecutar un máximo de 10^6 instrucciones por segundo. Por término medio, las instrucciones necesitan cinco ciclos máquina, tres de los cuales utilizan el bus de memoria. Una operación de lectura o escritura en memoria utiliza un ciclo máquina. Suponga que el procesador se encuentra ejecutando continuamente programas en segundo plano (*background*) que requieren el 95 por ciento de la velocidad de ejecución de sus instrucciones pero ninguna instrucción de E/S. Asuma que un ciclo del procesador es igual a un ciclo del bus. En un momento dado el dispositivo de E/S se utiliza para transferir bloques muy grandes de datos entre la memoria principal M y D.

(a) Si se utiliza la E/S programada y cada transferencia de una palabra requiere que el procesador ejecute dos instrucciones, estime la máxima velocidad (en palabras por segundo) de transferencia de datos de E/S, posible a través de D.

(b) Estime la misma magnitud si se utiliza DMA.

7.19. Una fuente de datos produce caracteres IRA de siete bits, y se añade un bit de paridad a cada uno. Obtenga la expresión de la máxima velocidad efectiva de transferencia de datos (velocidad de bits de datos IRA) en una línea de R bps para las situaciones siguientes:

(a) Transmisión asíncrona, con 1.5 bits de parada.

(b) Transmisión síncrona de bit, con una trama formada por 48 bits de control y 128 bits de información.

(c) Igual que (b), pero con un campo de información de 1024 bits.

(d) Transmisión síncrona de carácter, con nueve caracteres de control por trama y 16 caracteres de información.

(e) Igual que (d), con 128 caracteres de información.

7.20. El siguiente problema se basa en la ilustración de un mecanismo de E/S que se sugiere en [ECKE90] (Figura 7.22):

Dos mujeres están cada una a un lado de una valla muy alta. Una de ellas, llamada Servidora-de-Manzanas, tiene en su lado de la valla un precioso manzano cargado de manzanas, y disfruta proporcionando manzanas a la otra cuando esta se las pide. A la otra mujer, llamada Comedora-de-Manzanas, le encanta comer manzanas pero no tiene ninguna. De hecho, debe comer manzanas con una frecuencia fija establecida (una manzana cada día, por prescripción facultativa). Si come manzanas con una frecuencia mayor, se volvería enferma. Si las come menos frecuentemente, sufriría anemia. Ninguna de las mujeres puede hablar, por lo que el problema consiste en que la Servidora-de-Manzanas proporcione manzanas a la Comedora-de-Manzanas con la velocidad correcta.

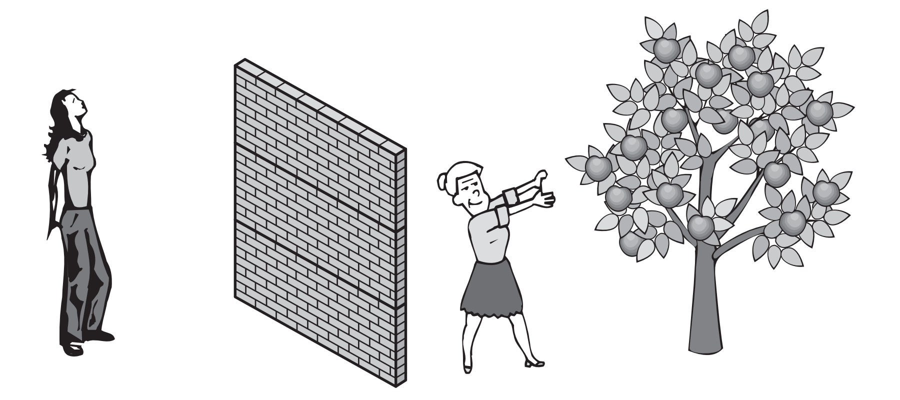


Figura 7.22. Un problema de manzanas.

(a) Asuma que hay un reloj despertador en lo alto de la valla, y que el despertador permite seleccionar varias horas de alarma. ¿Cómo puede utilizarse el reloj para resolver el problema? Dibuje un diagrama de tiempos para ilustrar la solución.

(b) Ahora asuma que no hay reloj despertador. En su lugar la Comedora-de-Manzanas tiene un banderín que puede agitar cada vez que necesite una manzana. Sugiera una nueva solución. ¿Serviría de algo que la Servidora-de-Manzanas tuviera otro banderín? Si es así, considérelo en la solución. Discuta las desventajas de esta posibilidad.

(c) Ahora olvide el banderín y asuma que se dispone de una cuerda suficientemente larga. Sugiera una solución mejor a la indicada en (b) utilizando la cuerda.

7.21. Asuma que un procesador de 16 bits y dos de 8 bits deben conectarse a un bus del sistema. Considere los siguientes detalles:

1. Todos los microprocesadores tienen el hardware necesario para cualquier tipo de transferencia: E/S programada, E/S mediante interrupciones y DMA.
2. Todos los microprocesadores tienen un bus de direcciones de 16 bits.
3. Hay dos tarjetas de memoria, de 64 Kbytes cada una, conectadas al bus. El diseñador desea que se comparta la mayor cantidad de memoria posible.
4. El bus del sistema permite un máximo de cuatro líneas de interrupción y una de DMA. Haga las suposiciones adicionales que necesite, y:

(a) Establezca las especificaciones del bus en términos del número y tipo de líneas.

(b) Describa un posible protocolo para la comunicación en el bus (es decir, las secuencias para la lectura/escritura, interrupción, y DMA).

(c) Explique cómo es la interfaz de los dispositivos indicados arriba para conectarse al bus.

CAPÍTULO 8

Sistemas operativos

ASPECTOS CLAVE

- El sistema operativo (SO) es el software que controla la ejecución de los programas en el procesador y gestiona sus recursos. Ciertas funciones del sistema operativo, como la planificación de procesos y la gestión de memoria, solo pueden realizarse eficaz y rápidamente si el procesador incluye ciertos elementos hardware que den soporte al sistema operativo. Prácticamente todos los procesadores disponen de dichos elementos en mayor o menor medida, incluyendo hardware para la gestión de la memoria virtual y para la gestión de procesos. Este hardware incluye registros y buffers de propósito específico, y circuitería para realizar tareas básicas de gestión de recursos.
- Una de las funciones más importantes del sistema operativo es la planificación de procesos o tareas. El sistema operativo determina qué proceso debe ejecutarse en cada momento. Usualmente, el hardware interrumpirá un proceso en ejecución en determinados instantes para permitir que el sistema operativo tome una nueva decisión de planificación de forma que el tiempo se reparta por igual entre los procesos.
- Otra función importante del sistema operativo es la gestión de memoria. La mayoría de los sistemas operativos actuales implementan la memoria virtual. Esta proporciona dos beneficios: (1) un proceso puede ejecutarse en memoria principal sin que todas sus instrucciones y datos se encuentren en memoria principal en un momento determinado, y (2) el espacio de memoria disponible para un programa puede exceder bastante del espacio existente en la memoria principal del sistema. Aunque el software es el encargado de la gestión de memoria, el sistema operativo aprovecha el soporte hardware que proporciona el procesador, que incluye el hardware para la paginación y la segmentación.

Aunque este texto se centra en el hardware del computador, hay un área del software que debe considerarse: el sistema operativo del computador. El sistema operativo es un programa que administra los recursos del computador, proporciona servicios a los programadores y planifica la ejecución de otros programas. Un cierto conocimiento de los sistemas operativos es esencial para entender los mecanismos mediante los que la CPU controla el computador. En particular, los efectos de las interrupciones y de la gestión de la jerarquía de memoria se explican mejor en este contexto.

El capítulo comienza con una revisión, una breve historia de los sistemas operativos. La mayor parte del capítulo considera las dos funciones del sistema operativo más relevantes para el estudio de la organización y la arquitectura del computador: la planificación y la gestión de memoria.

8.1. CONCEPTOS BÁSICOS SOBRE SISTEMAS OPERATIVOS

OBJETIVOS Y FUNCIONES DEL SISTEMA OPERATIVO

Un sistema operativo es un programa que controla la ejecución de los programas de aplicación y actúa como interfaz entre el usuario y el hardware del computador. Se puede considerar que un sistema operativo tiene dos objetivos:

- **Comodidad:** un sistema operativo hace que un computador sea más fácil y cómodo de usar.
- **Eficiencia:** un sistema operativo permite que los recursos del computador se utilicen de forma eficiente.

Examinemos uno por uno estos dos aspectos del sistema operativo.

El sistema operativo como una interfaz usuario/computador. El hardware y el software utilizado por las aplicaciones de usuario pueden verse como una jerarquía o serie de capas, tal y como se representa en la Figura 8.1. El usuario de las aplicaciones se denomina usuario final y generalmente no conoce la arquitectura del computador. Así, el usuario final tiene una visión del computador en términos de una aplicación. Esta aplicación puede utilizarse mediante un lenguaje de programación y ha sido desarrollada por un programador de aplicaciones. Resulta evidente que si los programas de aplicación se tuvieran que desarrollar en términos del repertorio de instrucciones máquina que son las que permiten el control directo del hardware del computador, la tarea sería de una complejidad abrumadora. Para facilitar el trabajo, existe un conjunto de programas del sistema. Algunos de

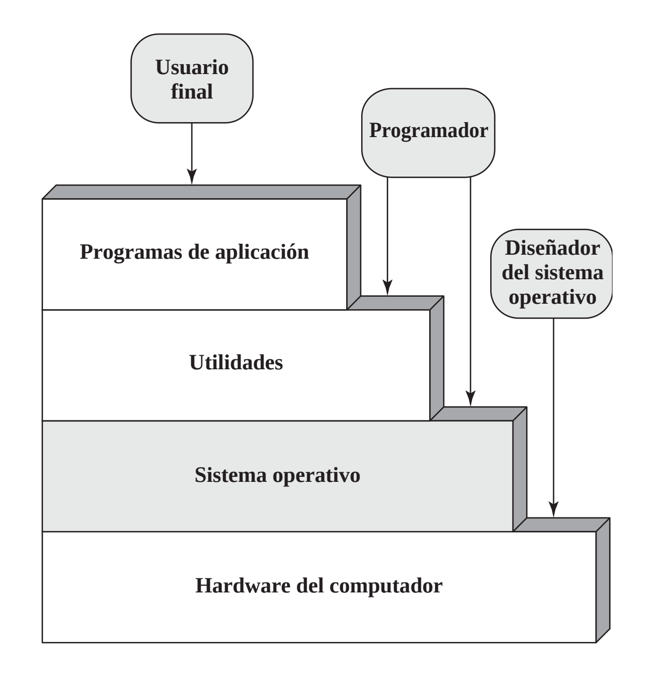


Figura 8.1. Capas y puntos de vista de un computador.

estos programas se denominan **utilidades**. Estas realizan funciones que se usan frecuentemente para ayudar en la elaboración de los programas, la gestión de los ficheros y el control de los dispositivos de E/S. Un programador hará uso de estos medios al desarrollar una aplicación, y la aplicación, mientras se está ejecutando, llamará a las utilidades para realizar ciertas funciones. El programa del sistema más importante es el sistema operativo. El sistema operativo oculta los detalles del hardware al programador y le proporciona una interfaz adecuada para utilizar el sistema. Actúa como mediador, facilitando al programador y a los programas de aplicación el acceso y el uso de los medios y servicios del sistema.

Resumiendo, el sistema operativo usualmente proporciona servicios en las siguientes áreas:

- **Creación de programas:** el sistema operativo proporciona cierta variedad de servicios y medios, tales como editores y depuradores, para ayudar al programador en la elaboración de programas. Usualmente, estos servicios son utilidades que propiamente no forman parte del sistema operativo, pero se accede a ellos a través de dicho sistema operativo**.**
- **Ejecución de programas:** para ejecutar un programa es preciso realizar una serie de tareas. Las instrucciones y los datos deben cargarse en memoria principal, los dispositivos de E/S y los ficheros deben iniciarse, y deben prepararse otros recursos. El sistema operativo proporciona todo eso al usuario.
- **Acceso a los dispositivos de E/S:** cada dispositivo de E/S necesita su conjunto particular de instrucciones y señales de control para poder operar. El sistema operativo se encarga de esos detalles para que el programador pueda pensar simplemente en términos de lecturas y escrituras.
- **Acceso controlado a los ficheros:** en el caso de ficheros, el control debe incluir el conocimiento no solo de la naturaleza del dispositivo (disco, cinta) sino también del formato del fichero y del medio de almacenamiento. Nuevamente, el sistema operativo se ocupa de los detalles. Es más, en el caso de un sistema con múltiples usuarios simultáneos, el sistema operativo puede proporcionar mecanismos de protección para controlar el acceso a los recursos compartidos, tales como los ficheros.
- **Acceso al sistema:** en el caso de un sistema compartido o público, el sistema operativo controla el acceso al sistema como un todo y a los recursos específicos del sistema. La función de acceso debe proporcionar protección de los recursos y datos frente a los usuarios no autorizados y debe resolver los conflictos por el acceso a los recursos compartidos.
- **Detección de errores y respuesta:** mientras el computador está funcionando pueden producirse diversos errores. Entre estos están los errores hardware internos y externos, tales como los errores de memoria o los fallos o comportamiento incorrecto de dispositivos; y errores diversos del software, tales como el desbordamiento (*overflow*) aritmético, el intento de acceder a una posición de memoria no permitida o la incapacidad del sistema operativo para responder una petición generada por una aplicación. En cada caso, el sistema operativo debe responder de forma que se supere la condición de error con el menor impacto para las aplicaciones que se están ejecutando. La respuesta del sistema operativo puede implicar abortar el programa que causó el error, reintentar la operación, o simplemente notificar el error a la aplicación.
- **Gestión de cuentas (*accounting*):** un buen sistema operativo debe almacenar la estadística de uso de los distintos recursos y supervisar los parámetros de prestaciones tales como el tiempo

de respuesta. En cualquier sistema, esta información es útil para anticipar la necesidad de futuras ampliaciones y ajustes que mejoren las prestaciones del sistema. En un sistema multiusuario, esta información puede utilizarse para facturar las cantidades que deben aportar los usuarios.

El sistema operativo como administrador de recursos. Un computador es un conjunto de recursos para transferir, almacenar y procesar datos, y para controlar esas funciones. El sistema operativo es responsable de la administración de esos recursos.

¿Es correcto decir que es el sistema operativo el que controla la transferencia, el almacenamiento, y el procesamiento de los datos? Desde un punto de vista, la respuesta es sí; al administrar los recursos del computador, el sistema operativo controla las funciones básicas del computador. Pero este control se ejerce de una forma curiosa. Normalmente, se piensa en un mecanismo de control como algo externo a aquello que se controla o, al menos, como algo que es una parte distinta y separada de lo que se controla (por ejemplo, un sistema de calefacción se controla mediante un termostato, que es algo completamente distinto al sistema de generación y distribución de calor). Este no es el caso del sistema operativo, que es un mecanismo de control inusual por dos razones:

- El sistema operativo funciona de la misma forma que el software ordinario del computador; esto es, se trata de un programa ejecutado por el procesador.
- El sistema operativo frecuentemente cede el control y depende del procesador para recuperar el control.

El sistema operativo, de hecho, no es nada más que un programa de computador. Como otros programas, proporciona instrucciones al procesador. La única diferencia se encuentra en el objetivo del programa. El sistema operativo dirige al procesador en el uso de otros recursos del sistema y en la temporización de la ejecución de otros programas. Pero para que el procesador pueda realizar esas cosas, debe dejar de ejecutar el sistema operativo y ejecutar otros programas. Así, el sistema operativo cede el control para que el procesador pueda realizar el trabajo «útil» y recupera el control posteriormente para preparar al procesador para el siguiente trozo de trabajo a realizar. Los mecanismos implicados en este proceso se aclararán a lo largo del capítulo.

La Figura 8.2 indica los principales recursos que administra el sistema operativo. Una parte del sistema operativo está en la memoria principal. Esta incluye el **núcleo (*kernel*),** que realiza las funciones más frecuentemente utilizadas por el sistema operativo y, en un momento dado, las otras partes del sistema operativo que están actualmente en uso. El resto de la memoria principal contene otros programas y datos. Como se verá, la asignación de este recurso (memoria principal) está controlada conjuntamente por el sistema operativo y el hardware de gestión de memoria del procesador. El sistema operativo decide cuándo un programa en ejecución puede usar un dispositivo de E/S y controla el acceso y el uso de los ficheros. El procesador es en sí un recurso, y el sistema operativo debe determinar el tiempo que el procesador dedica a la ejecución de cada programa. En el caso de un sistema multiprocesador, esta decisión debe incluir a todos los procesadores.

TIPOS DE SISTEMAS OPERATIVOS

Para distinguir entre los distintos tipos de sistemas operativos existen ciertas características clave. Las características se agrupan en dos dimensiones distintas. La primera dimensión especifica si se

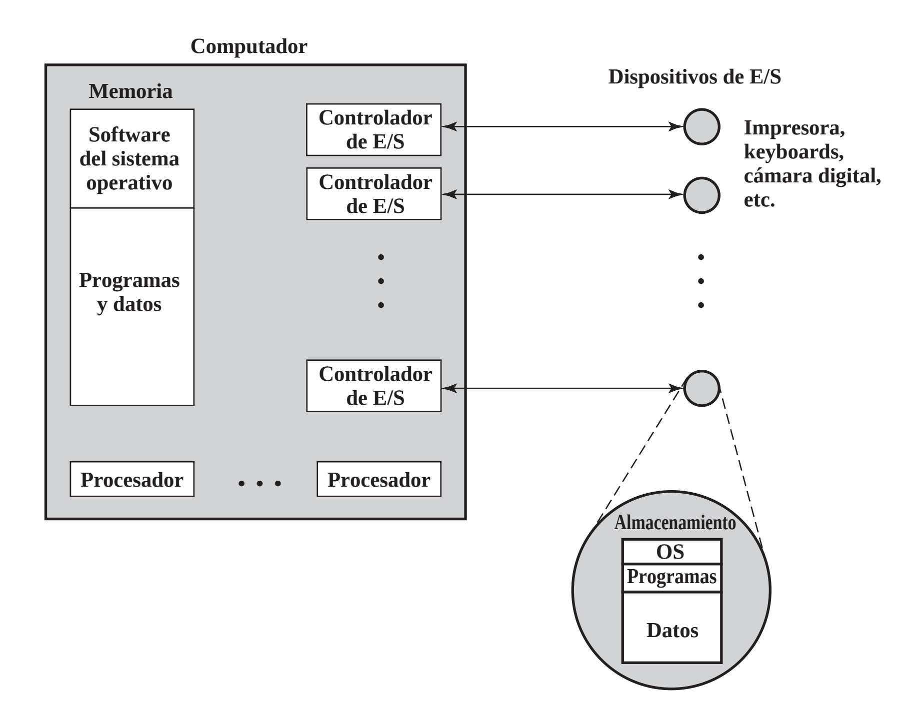


Figura 8.2. El sistema operativo como gestor de recursos.

trata de un sistema de colas (*batch*) o interactivo. En un sistema **interactivo**, el usuario/programador interactúa directamente con el computador, usualmente a través de un terminal de pantalla y teclado, para solicitar la ejecución de un trabajo o realizar una transacción. Además, el usuario puede, según la naturaleza de la aplicación, comunicarse con el computador durante la ejecución del trabajo. Un sistema **de colas** es lo opuesto a uno interactivo. El programa de usuario se introduce en una cola junto con programas de otros usuarios. Después de que el programa haya terminado, los resultados se proporcionan al usuario. Actualmente es raro encontrar sistemas de colas puros. Sin embargo, resulta útil para la descripción de los sistemas operativos contemporáneos examinar brevemente los sistemas de colas.

Otra dimensión independiente especifica si el sistema utiliza **multiprogramación** o no. Con la multiprogramación se intenta mantener el procesador ocupado tanto como sea posible, haciéndolo trabajar en más de un programa al mismo tiempo. Varios programas se cargan en la memoria, y el procesador conmuta rápidamente entre ellos. La alternativa es un sistema de **monoprogramación** que trabaja solo en un programa en cada momento.

Los primeros sistemas. En los primeros computadores desde el final de la década de los cuarenta a la mitad de la de los cincuenta, el programador interactuaba directamente con el hardware del computador. Estas máquinas se accionaban desde una consola, constituida por luces indicadoras, interruptores, algún dispositivo de entrada y una impresora. Los programas en código máquina se cargaban mediante el dispositivo de entrada (por ejemplo, un lector de tarjetas). Si un error hacía detenerse al programa, las luces indicaban la condición de error. El programador debe proceder a comprobar los registros y la memoria principal para determinar la causa del error. Si el programa termina, la salida aparecerá en la impresora.

Estos primeros sistemas presentaban dos problemas fundamentales:

- **Planificación:** la mayoría de las instalaciones utilizaban una lista para reservar tiempo en la máquina. Un usuario normalmente podía reservarse espacios de tiempo múltiplos de media hora. Sin embargo, podía haber reservado una hora y en cambio terminar en 45 minutos; lo que ocasionaba un tiempo desperdiciado en el que el computador estaba parado. Por otra parte, el usuario podía tener problemas al ejecutar el programa, no terminar en el tiempo asignado y verse forzado a parar sin resolver el problema.
- **Tiempo de preparación:** un único programa, llamado **trabajo (*job*)**, se encargaba de cargar en memoria el compilador y el programa en lenguaje de alto nivel (programa fuente), guardar el programa compilado (programa objeto) y después cargar y enlazar juntos el programa objeto y las funciones comunes. Cada uno de estos pasos podía implicar montar y desmontar cintas o activar terminales de tarjetas. Por tanto, se consumía una considerable cantidad de tiempo solo en preparar el programa para que se pudiera ejecutar.

Este modo de funcionamiento podría llamarse procesamiento en serie, reflejando el hecho de que los usuarios acceden en serie al compuador. Con el tiempo se han ido desarrollando diversas herramientas integrantes del software del sistema que proporcionaban un procesamiento en serie más eficiente. Entre estas están las bibliotecas de funciones usuales, enlazadores, cargadores, depuradores y rutinas de control de E/S, de las que todos los usuarios pueden disponer.

Sistemas de colas simples. Las primeras máquinas eran muy caras, y por ello era muy importante maximizar la utilización de la máquina. El tiempo perdido debido a la planificación y a la preparación era inaceptable.

Para mejorar la utilización, se desarrollaron los sistemas de colas sencillos. Con un sistema de este tipo, llamado **monitor**, el usuario ya no tiene acceso directo a la máquina. En cambio, el usuario envía el trabajo, en tarjetas o en cinta, a un operador del computador que pone los trabajos en cola y sitúa toda la cola en un dispositivo de entrada al que accede el monitor.

Para comprender cómo trabaja el esquema, considerémoslo desde dos puntos de vista: el del monitor y el del procesador. Desde el punto de vista del monitor, es él el que controla la secuencia de eventos. Para que esto sea así, el monitor está siempre en la memoria principal y dispuesto para ejecutarse (Figura 8.3). Esta parte se denomina **monitor residente**. El resto del monitor consiste en utilidades y funciones comunes que son cargadas como subrutinas del programa de usuario al iniciarse cualquier trabajo que las necesite. El monitor introduce uno a uno los trabajos desde el dispositivo de entrada (usualmente un lector de tarjetas o de cintas magnéticas). A medida que es leído, el trabajo en cuestión se sitúa en el área de programas de usuario y se cede el control a dicho trabajo. Cuando el trabajo termina, se devuelve el control al monitor, que inmediatamente lee el siguiente trabajo. Los resultados de cada trabajo se imprimen para que el usuario pueda disponer de ellos.

Ahora consideremos esta secuencia desde el punto de vista del procesador. En cierto instante de tiempo, el procesador está ejecutando instrucciones captadas de la porción de memoria que contiene al monitor. Estas instrucciones hacen que se lea el siguiente trabajo y se pase a otra zona de memoria principal. Una vez que se ha introducido el trabajo, el procesador ejecutará una instrucción de salto del monitor que hace que el procesador prosiga la ejecución en la posición de memoria correspondiente al comienzo del programa de usuario. El procesador ejecutará entonces las instrucciones del programa de usuario hasta que encuentre una condición de final o de error. En cualquiera de los casos

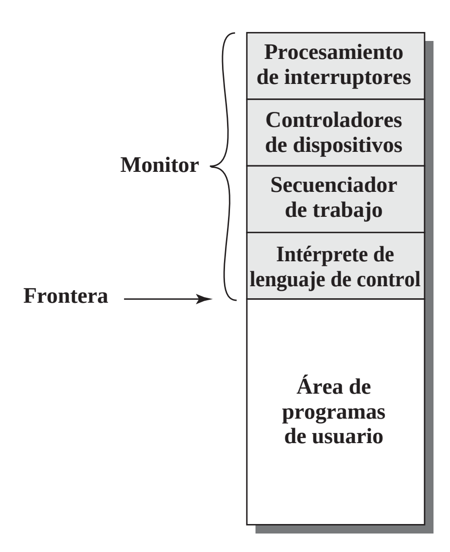


Figura 8.3. Distribución de la memoria para un monitor residente.

el procesador capta la siguiente instrucción e ejecutar del programa monitor. Así, la frase «el control pasa a un trabajo» simplemente significa que el procesador está captando y ejecutando instrucciones de un programa de usuario, y «el control se devuelve al monitor» significa que el procesador está captando y ejecutando instrucciones del programa monitor.

Queda claro que el monitor resuelve el problema de la planificación. Al existir una serie de trabajos en cola, se pueden ejecutar tan rápido como es posible sin que haya tiempos muertos.

¿Qué pasa con el problema del tiempo de preparación? El monitor lo resuelve también. Dentro de cada trabajo se incluyen instrucciones en un *lenguaje de control de trabajos (*JCL, *Job Control Language)*. Se trata de un tipo especial de lenguaje de programación utilizado para dar las instrucciones al monitor. Un ejemplo sencillo es aquel en el que el usuario está enviando un programa escrito en FORTRAN más algunos datos que se utilizan en el programa. Cada instrucción FORTRAN y cada dato se encuetra en una tarjeta perforada distinta o en un registro diferente de una cinta magnética. Además de las líneas FORTRAN y de datos, el trabajo incluye las instrucciones de control de trabajo, que se distinguen porque empiezan con «$». El formato del trabajo podría ser:

```
$JOB
$FTN
 ⋮          } Instrucciones FORTRAN
$LOAD
$RUN
 ⋮          } Data
$END
```

Para ejecutar este trabajo, el monitor lee la línea $FTN y carga el compilador apropiado desde el dispositivo de almacenamiento masivo (usualmente una cinta). El compilador traduce el programa de

usuario a código objeto, que se almacena en memoria o en el dispositivo de almacenamiento masivo. Si se almacena en memoria, la operación se denomina «compilar, cargar, y ejecutar». Si se almacena en cinta magnética, entonces es necesario utilizar la instrucción $LOAD. Esta instrucción es leída por el monitor, que vuelve a tomar el control tras la compilación. El monitor llama al cargador, que sitúa el programa en memoria en lugar del compilador y le transfiere el control. De esta forma, una gran parte de la memoria principal puede compartirse por subsistemas diferentes, aunque solo uno de ellos puede estar residente y ejecutándose en cada instante.

El monitor, o sistema operativo de colas, es simplemente un programa de computador. Se basa en la posibilidad que tiene el procesador de captar instrucciones de diferentes zonas de la memoria principal para tomar y ceder el control. Además se necesita que el hardware proporcione ciertas funciones:

- **Protección de memoria:** mientras el programa de usuario se está ejecutando, no debe alterarse el área de memoria que contiene al monitor. Si se intenta, el hardware del procesador detecta un error y transfiere el control al monitor. El monitor aborta el trabajo, imprime un mensaje de error, y carga el siguiente trabajo.
- **Temporización:** se debe utilizar un temporizador para evitar que un único trabajo monopolice el uso del sistema. El temporizador se actualiza al comienzo de cada trabajo. Si el tiempo termina, se produce una interrupción, y el control vuelve al monitor.
- **Instrucciones privilegiadas:** ciertas instrucciones que se denominan privilegiadas solo pueden ser ejecutadas por el monitor. Entre estas están las instrucciones de E/S para que el monitor tenga el control de todos los dispositivos de E/S. Esto impide, por ejemplo, que un programa de usuario lea accidentalmente las instrucciones del siguiente trabajo. Si un programa de usuario desea realizar una E/S, debe solicitar al monitor que realice la operación por él. Si el procesador encuentra una instrucción privilegiada mientras ejecuta un programa de usuario, el hardware del procesador lo considera un error y transfiere el control al monitor.
- **Interrupciones**: los primeros modelos de computadores no disponian de esta capacidad. Esta característica proporciona al procesador más flexibilidad para ceder y recuperar el control de los programas de usuario.

El tiempo del procesador se alterna entre la ejecución de los programas de usuario y la ejecución del monitor. Se han sacrificado dos cosas: parte de la memoria principal está ocupada por el monitor, y parte del tiempo de la máquina es consumido por el monitor. Ambas cosas constituyen una cierta penalización (*overhead*). Incluso con esta penalización, los sistemas de colas sencillos mejoran la utilización del computador.

Sistemas de colas multiprogramados. Incluso con la sucesión automática de trabajos que proporcionan los sistemas de colas sencillos, el procesador está parado a menudo. El problema surge porque los dispositivos de E/S son lentos en comparación con el procesador. La Figura 8.4 describe una situación típica. El cálculo se refiere a un programa que procesa un fichero de registros y ejecuta, por término medio, cien instrucciones máquina por registro. En este ejemplo el computador pasa ¡alrededor del 96 por ciento de su tiempo esperando que los dispositivos de E/S terminen de transferir datos! La Figura 8.5a ilustra esta situación. El procesador consume cierto tiempo ejecutando instrucciones hasta que llega a una instrucción de E/S. Entonces debe esperar hasta que esa instrucción de E/S concluya para continuar.

Leer un registro	0,0015 segundos
Ejecutar 100 instrucciones	0,0001 segundos
Escribir un registro	0,0015 segundos
TOTAL	0,0031 segundos

$$\text{Porcentaje de uso de la CPU} = \frac{1}{31} = 0{,}032 = 3{,}2\%$$

Figura 8.4. Ejemplo de utilización de un sistema.

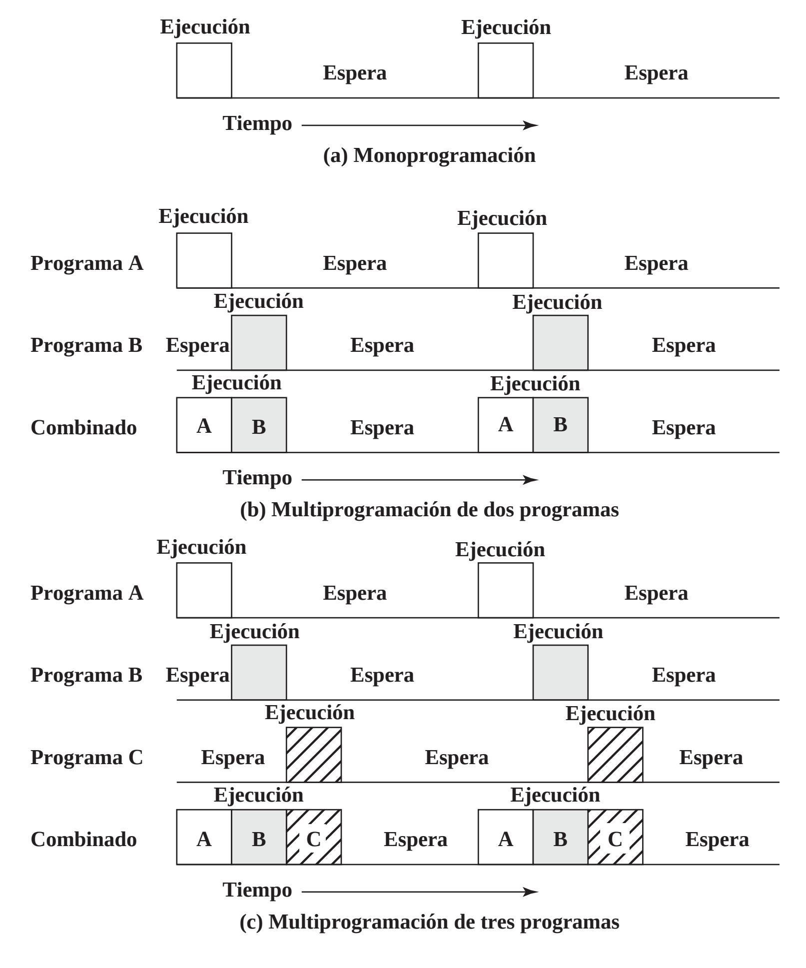


Figura 8.5. Ejemplo de multiprogramación.

Esta ineficiencia se puede evitar. Se ha indicado que debe haber memoria suficiente para dar cabida al sistema operativo (monitor residente) y a un programa de usuario. Supóngase que hay sitio para el sistema operativo y dos programas de usuario. Ahora, cuando un trabajo necesita esperar debido a una E/S, el procesador puede conmutar al otro trabajo, que posiblemente no estará esperando una E/S

(Figura 8.5b). Es más, se podría expandir la memoria para disponer de tres, cuatro o más programas entre los que conmutar (Figura 8.5c). Este proceso se conoce como **multiprogramación** o **multitarea (*multitasking*)** [1]. Es el tema central de los sistemas operativos modernos.

Ejemplo 8.1. Este ejemplo ilustra el beneficio de la multiprogramación. Considérese un computador con una memoria disponible (no utilizada por el sistema operativo) de 250 Mb, un disco, un terminal y una impresora. Se envían al mismo tiempo tres programas, Trabajo1, Trabajo2 y Trabajo3, para su ejecución. Sus atributos se enumeran en la Tabla 8.1. Asumimos requisitos mínimos de procesador para Trabajo2 y Trabajo3, y un uso contínuo del disco y la impresora por parte de Trabajo3. En un entorno de colas simple, estos trabajos se ejecutarían sucesivamente uno tras otro. Así, Trabajo1 termina en cinco minutos. Trabajo2 debe esperar a que los cinco minutos hayan pasado, y termina quince minutos después. Trabajo3 empieza después de veinte minutos y termina treinta minutos después de que se enviara. La utilización media de los recursos, el rendimiento, y los tiempos de respuesta se muestran en la columna de monoprogramación de la Tabla 8.2. La utilización dispositivo por dispositivo se ilustra en la Figura 8.6a. Es evidente que hay una importante infrautilización de todos los recursos cuando se promedia su uso en el periodo de tiempo de 30 minutos.

Ahora suponga que los trabajos se ejecutan concurrentemente bajo un sistema operativo con multiprogramación. Puesto que hay poca competencia entre los trabajos por los recursos, los tres pueden ejecutarse en un tiempo casi mínimo al coexistir en el computador con el resto (asumiendo que se asigna a Trabajo2 y Trabajo3 tiempo de procesador suficiente para mantener activas sus operaciones de entrada y salida). El Trabajo1 todavía necesitará cinco minutos para terminar, pero al final de ese tiempo Trabajo2 se habrá completado en un tercio y Trabajo3 en la mitad. Los tres trabajos habrán terminado en un tiempo de quince minutos. La mejora es evidente si se examina la columna de multiprogramación de la Tabla 8.2, obtenida a partir del histograma de la Figura 8.6b

Tabla 8.1. Atributos de ejecución de un programa ejemplo.

	Trabajo 1	**Trabajo 2**	**Trabajo 3**
Tipo de trabajo	Muchos cálculos	Muchas E/S	Muchas E/S
Duración	5 min.	15 min.	10 min.
Memoria necesaria	50 M	100 M	80 M
¿Necesita disco?	No	No	Sí
¿Necesita terminal?	No	Sí	No
¿Necesita impresora?	No	No	Sí

[1] El término *multitarea* se reserva en algunos casos para hacer referencia a varias tareas dentro de un mismo programa que pueden gestionarse concurrentemente por sistema operativo, a diferencia de *multiprogramación*, que se referiría en ese caso a varios procesos de programas diferentes. No obstante, es más común equiparar los términos *multitarea* y *multiprogramación*, tal y como se hace en la mayoría de los diccionarios estándar (por ejemplo, el IEEE Std 100-1992: *The New IEEE Standard Dictionary of Electrical and Electronics Terms*).

Tabla 8.2. Efectos de la multiprogramación sobre la utilización de recursos.

	Trabajo 1	Trabajo 2
Utilización del procesador	20%	40%
Utilización de la memoria	33%	67%
Utilización del disco	33%	67%
Utilización de la impresora	33%	67%
Tiempo transcurrido	30 min.	15 min.
Rendimiento	6 trabajos/hora	12 trabajos/hora
Tiempo de respuesta medio	18 min.	10 min.

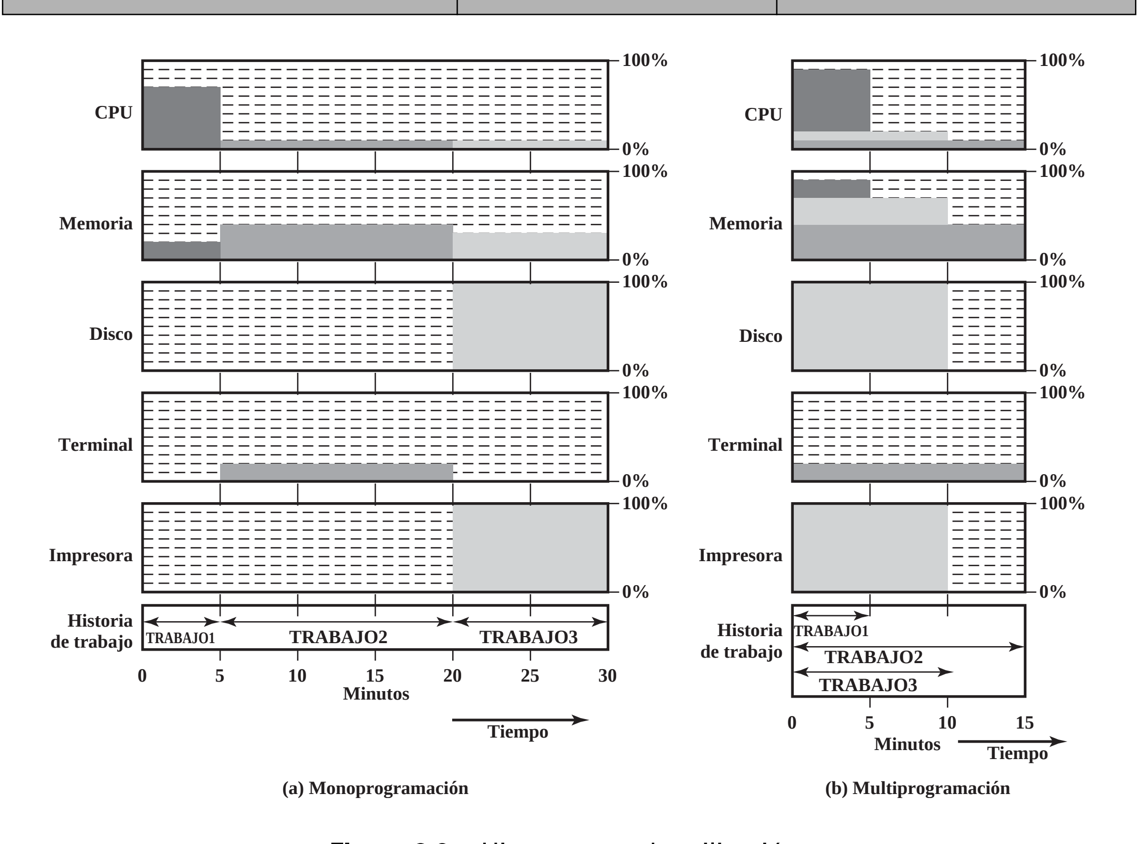


Figura 8.6. Histogramas de utilización.

Igual que en un sistema de colas sencillo, un sistema de colas multiprogramado es un programa que se apoya en ciertas características del hardware del computador. La característica más notable de utilidad para la multiprogramación es el soporte hardware para las interrupciones y el DMA. Con las E/S mediante interrupciones o mediante DMA, la CPU puede lanzar una orden de E/S para un trabajo y continuar ejecutando otro trabajo mientras el controlador de dispositivo se encarga de realizar la E/S. Cuando se completa la operación de E/S, la CPU es interrumpida y el control pasa a un programa de gestión de interrupciones del sistema operativo. Entonces, el sistema operativo pasa el control a otro trabajo.

Los sistemas operativos multiprogramados son bastante sofisticados en comparación con los sistemas de un solo programa o **monoprogramados**. Para tener varios trabajos listos para ejecutarse, deben mantenerse en memoria, precisándose una cierta **gestión de la memoria**. Además, si varios trabajos están listos para ejecutarse, el procesador debe decidir cuál de ellos se ejecuta, lo que implica utilizar algún algoritmo de planificación. Estos conceptos se discuten más adelante en este capítulo.

Sistemas de tiempo compartido. Con el uso de la multiprogramación, el procesamiento en colas puede ser bastante eficiente. Sin embargo, para muchos trabajos es deseable disponer de un modo en el cual el usuario interactúe directamente con el computador. De hecho, para algunos trabajos, tales como el procesamiento de transacciones, es esencial el modo interactivo.

Hoy en día, los requisitos para el procesamiento interactivo pueden ser, y a menudo son, satisfechos por un microcomputador. Esta opción no era posible en los sesenta, cuando la mayoría de los computadores eran grandes y costosos. En su lugar, se desarrolló el tiempo compartido.

Igual que la multiprogramación permite que el procesador ejecute varios trabajos de la cola en un intervalo de tiempo, también se puede hacer que ejecute varios trabajos interactivos. En este caso, la técnica se denomina tiempo compartido, puesto que el tiempo del procesador se comparte entre varios usuarios. En un sistema de tiempo compartido, varios usuarios acceden simultáneamente al sistema a través de terminales, mientras que el sistema operativo alterna la ejecución de fragmentos o ráfagas de cómputo correspondientes a cada usuario. Así, si hay n usuarios que solicitan servicio al mismo tiempo, cada usuario solo aprovechará, por término medio, una fracción igual a $1/n$ de la velocidad efectiva del procesador, y eso sin contar el tiempo dedicado al sistema operativo. No obstante, dado el relativamente elevado tiempo de reacción humano, el tiempo de respuesta de un sistema diseñado correctamente debería ser comparable al que proporciona un computador dedicado.

Tanto las colas multiprogramadas como el tiempo compartido usan multiprogramación. Las diferencias esenciales se enumeran en la Tabla 8.3.

Tabla 8.3. Multiprogramación con colas frente a tiempo compartido.

	Multiprogramación con colas	**Tiempo compartido**
Objetivo principal	Maximizar la utilización del Procesador	Minimizar el tiempo de respuesta
Fuente de instrucciones para el sistema operativo	Instrucciones de un lenguaje de control de trabajos que proporciona el propio trabajo	Órdenes introducidas a través del terminal

8.2. PLANIFICACIÓN

La clave de la multiprogramación es la planificación. De hecho, usualmente implica tres tipos de planificación (Tabla 8.4). Las describiremos aquí. Pero primero, introduciremos el concepto de **proceso**. Este término fue utilizado por primera vez por los diseñadores de Multics en los sesenta. En cierta forma, se trata de un término más general que *trabajo*. Se han dado muchas definiciones del término *proceso*, entre ellas:

Tabla 8.4. Tipos de planificación.

Planificación a largo plazo	Decide si se añade al conjunto de programas a ser ejecutados
Planificación a medio plazo	Decide si se añade al número de procesos que están total o parcialmente en memoria principal
Planificación a corto plazo	Decide cuál de los procesos disponibles ejecutará el procesador
Planificación de E/S	Decide el procesador cuya solicitud de E/S pendiente se va a atender por parte del dispositivo de E/S disponible

- Un programa en ejecución.
- El «espíritu animado» de un programa.
- Aquella entidad a la que se asigna un procesador.

El concepto se aclarará a medida que avancemos.

PLANIFICACIÓN A LARGO PLAZO

El planificador a largo plazo determina qué programas se admiten para ser procesados en el sistema. De esta manera, este planificador controla el grado de multiprogramación (número de procesos en memoria). Una vez admitido, un trabajo o programa de usuario pasa a ser un proceso y se añade a una cola asociada al planificador a corto plazo. En algunos sistemas, un proceso nuevo comienza a partir de una sustitución en el intercambio (*swapping*), en cuyo caso se añade a la cola del planificador a medio plazo.

En un sistema de colas, o en la parte de colas de un sistema operativo de uso general, los trabajos nuevos que se envian pasan al disco y se mantienen en una cola. El planificador a largo plazo selecciona trabajos de esta cola cuando puede. Esto implica tomar dos decisiones. Primera, el planificador debe decidir si el sistema operativo puede aceptar uno o más procesos adicionales. Segunda, el planificador debe decidir qué trabajo o trabajos acepta y transforma en procesos. Los criterios que se utilizan deben incluir la prioridad, el tiempo de ejecución esperado y las E/S que se requieren.

Para los programas interactivos en un sistema de tiempo compartido, se genera una solicitud de proceso cuando un usuario intenta conectarse al sistema. Los usuarios en tiempo compartido no se introducen en una cola para mantenerse esperando a que el sistema los acepte. Por el contrario, el sistema operativo aceptará todos los usuarios autorizados hasta que el sistema se sature, según un criterio de saturación predefinido. En ese momento, si se produce una solicitud de conexión se responde con un mensaje que indica que el sistema está completo y el usuario debe intentar la conexión de nuevo, pasado un cierto tiempo.

PLANIFICACIÓN A MEDIO PLAZO

La planificación a medio plazo es parte de la función de intercambio, descrita en la Sección 8.3. Usualmente, la decisión de intercambiar un proceso se toma en función del grado de multiprogramación

que se desea mantener. En un sistema que no utilice memoria virtual, la gestión de la memoria también debe considerarse por el planificador a medio plazo, y en las decisiones tomadas en el intercambio deben tenerse en cuenta las necesidades de memoria de los procesos intercambiados.

PLANIFICACIÓN A CORTO PLAZO

El planificador a largo plazo se ejecuta de manera relativamente poco frecuente y toma las decisiones más genéricas de si aceptar un nuevo proceso o no, y qué proceso aceptar. El planificador a corto plazo, conocido también como **distribuidor (*dispatcher*)**, se ejecuta frecuentemente y toma la decisión más específica de qué trabajo se ejecuta a continuación.

Estados de los procesos. Para comprender el funcionamiento del planificador a corto plazo, necesitamos considerar el concepto de estado de un proceso. Durante el tiempo de vida de un proceso, la situación en que se encuentra cambiará un cierto número de veces. Su situación en cada instante de tiempo se denomina *estado*. El término *estado* se utiliza porque tiene la connotación de que existe cierta información que define la situación en que se encuentra el proceso en ese momento. Usualmente, se definen cinco estados para un proceso (Figura 8.7):

- **Nuevo (*New*):** el planificador de alto nivel admite un programa pero todavía no está preparado para ejecutarse. El sistema operativo iniciará el proceso, pasándolo al estado preparado.
- **Preparado (*Ready*):** el proceso está preparado para ejecutarse y se encuentra esperando acceso al procesador.
- **En ejecución (*Running*):** el proceso está siendo ejecutado por el procesador.
- **En espera (*Waiting*):** el proceso ha suspendido su ejecución al estar esperando algún recurso del sistema, tal como una E/S.
- **Parado (*Halted*):** el proceso ha terminado y será eliminado por el sistema operativo.

Para cada proceso del sistema, el sistema operativo debe mantener información de su estado indicando la situación en que se encuentra el proceso y cualquier información adicional necesaria para la

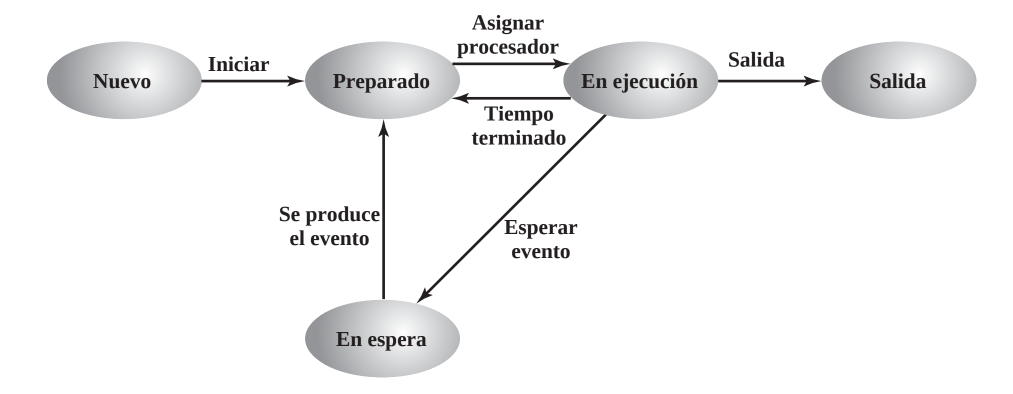


Figura 8.7. Modelo de proceso de cinco estados.

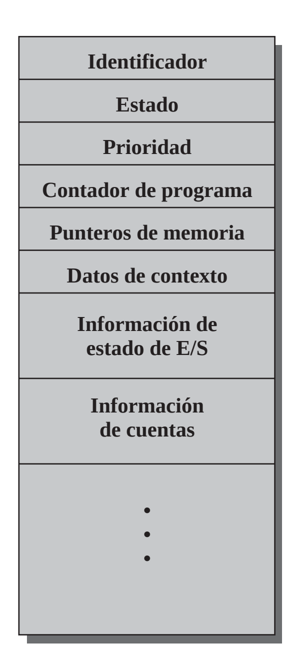


Figura 8.8. Bloque de control de procesos.

ejecución del mismo. Para eso, cada proceso se representa en el sistema operativo mediante un **bloque de control de proceso** (Figura 8.8) que usualmente está constituido por:

- **Identificador:** cada proceso en curso tiene un identificador único.
- **Estado:** el estado actual del proceso (Nuevo, Preparado, etc.).
- **Prioridad:** el nivel de prioridad relativo.
- **Contador de programa:** la dirección de la siguiente instrucción del programa a ejecutar.
- **Punteros a memoria:** las posiciones de memoria de inicio y final del proceso.
- **Datos de contexto:** son los datos de los registros del procesador cuando el proceso se está ejecutando, y se discutirán en la Parte Tercera. Por ahora, es suficiente decir que estos datos representan el «contexto» del proceso. El contexto junto con el contador de programa se guarda cuando el procesador abandona el estado Preparado. El procesador los recupera cuando reanuda la ejecución del proceso.
- **Información de estado de las E/S:** incluye las solicitudes de E/S pendientes, los dispositivos de E/S (por ejemplo, cintas) asignados al proceso, la lista de ficheros asignados al proceso, etc.
- **Información para contabilidad:** puede incluir el tiempo total y el tiempo de procesador utilizados, los límites de tiempo, los datos de las cuentas, etc.

Cuando el planificador acepta un nuevo trabajo o solicitud de ejecución de un usuario, crea un bloque de control de procesos en blanco y sitúa en él al proceso asociado en el estado Nuevo. Después de que el sistema haya completado correctamente el bloque de control de proceso, el proceso se transfiere al estado Preparado.

Técnicas de planificación. Para entender cómo el sistema operativo realiza la planificación de los trabajos en memoria, empezaremos considerando el ejemplo de la Figura 8.9. La figura muestra cómo se divide la memoria principal en un instante de tiempo dado. El núcleo del sistema operativo, por supuesto, siempre está residente. Además, hay un cierto número de procesos activos, por ejemplo A y B, a cada uno de los cuales se les asigna una porción de memoria.

Empezamos en un instante de tiempo dado cuando el proceso A está ejecutándose. El procesador toma las instrucciones del programa contenido en la partición de memoria de A. En un instante posterior, el procesador deja de ejecutar instrucciones de A y empieza a ejecutar instrucciones del área del sistema operativo. Esto puede suceder debido a una de estas tres razones:

1. El proceso A genera una llamada a un servicio (por ejemplo, una solicitud de E/S) del sistema operativo. La ejecución de A se suspende hasta que el sistema operativo ha completado el servicio solicitado.
2. El proceso A origina una *interrupción*. Una interrupción es una señal generada por el hardware que se envía al procesador. Cuando se detecta la señal, el procesador deja de ejecutar A

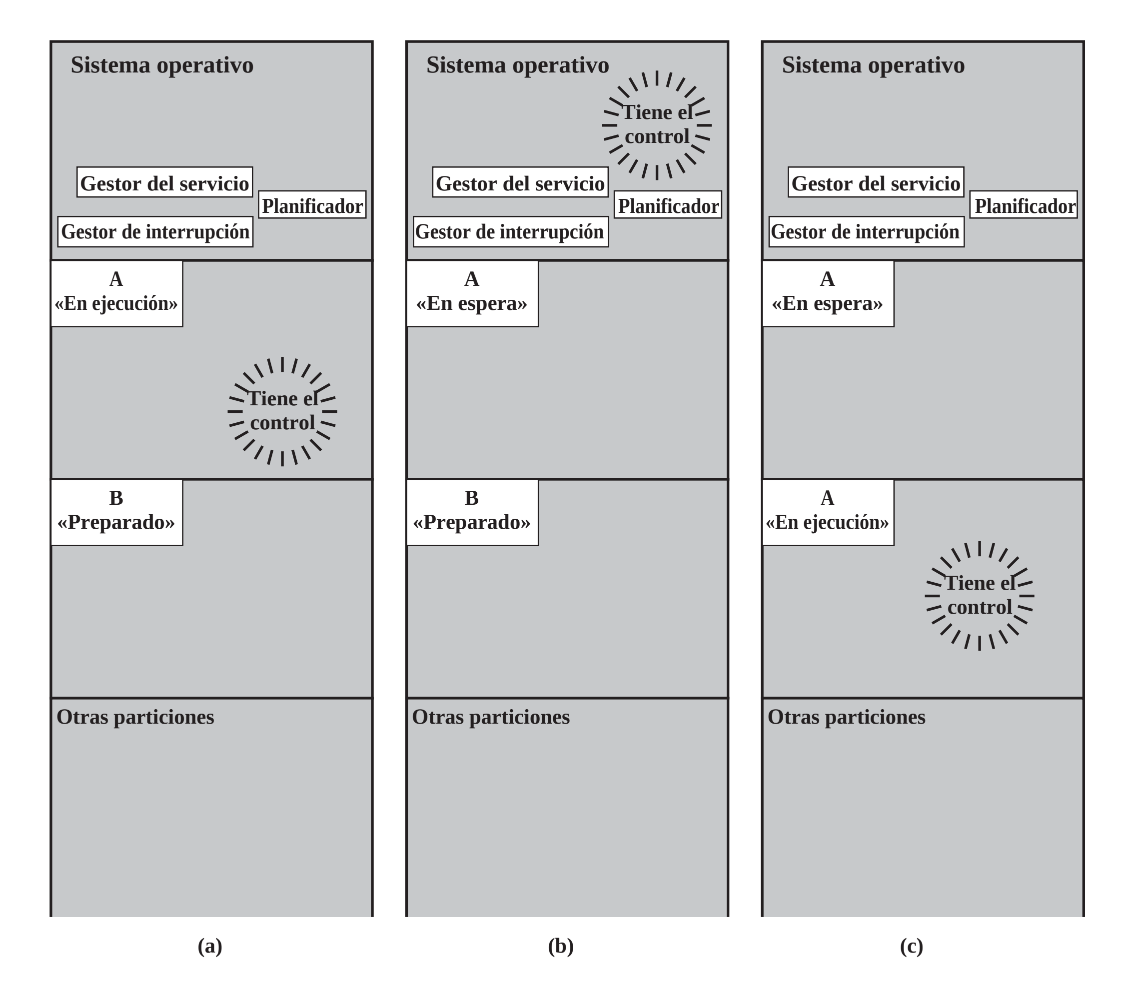


Figura 8.9. Ejemplo de planificación.

y pasa al gestor de interrupciones incluido en el sistema operativo. Hay una diversidad de eventos de A que pueden ocasionar la interrupción. Por ejemplo, un error tal como el intento de ejecutar una instrucción privilegiada. También se genera una interrupción cuando se agota el tiempo asignado al proceso. Para evitar que monopolice al procesador, cada proceso dispone del procesador solo durante un corto periodo de tiempo.

3. Algún hecho no relacionado con el proceso A que requiere atención origina una interrupción. Por ejemplo cuando se completa una operación de E/S.

En cualquier caso, el resultado es el siguiente. El procesador guarda los datos del contexto actual y el contador de programa de A en el bloque de control del proceso A y empieza a ejecutar el sistema operativo. El sistema operativo puede realizar alguna actividad, tal como iniciar una operación de E/S. Entonces, la porción del sistema operativo correspondiente al planificador a corto plazo decide el proceso que se ejecuta a continuación. En este ejemplo, se elige B. El sistema operativo hace que se restauren en el procesador los datos del contexto de B y se prosigue con la ejecución de B donde se dejó.

Este sencillo ejemplo aclara el funcionamiento básico del planificador a corto plazo. La Figura 8.10 muestra los elementos del sistema operativo que intervienen de manera más importante en la multiprogramación y en la planificación de procesos. El sistema operativo recibe el control del procesador al ejecutarse el gestor de interrupciones si se produce una interrupción, y al ejecutarse el gestor de llamadas de servicio si se solicita un servicio. Una vez se ha servido la llamada o la interrupción, vuelve a intervenir el planificador a corto plazo que selecciona un proceso para su ejecución.

Para realizar este trabajo, el sistema operativo utiliza un cierto número de colas. Cada cola es simplemente una lista de espera de procesos que necesitan un recurso. La **cola a largo plazo** es una lista de trabajos que esperan utilizar el sistema. Cuando las condiciones lo permitan, el planificador a largo plazo asignará memoria y creará un proceso para uno de los elementos que esperan en la cola. La **cola**

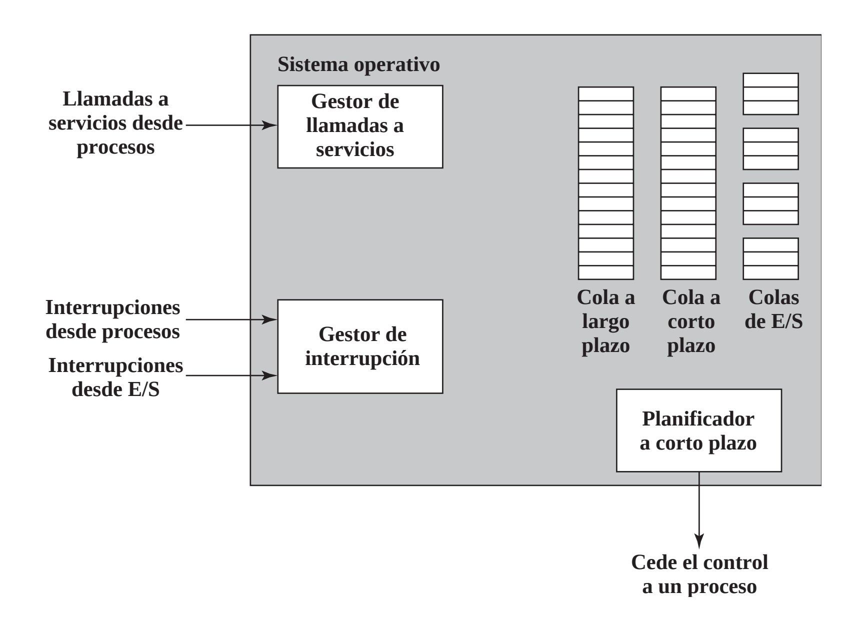


Figura 8.10. Elementos clave de un sistema operativo con planificación.

a corto plazo contiene a los procesos que se encuentran en estado Preparado. Cada uno de estos procesos podría ser el siguiente en utilizar el procesador. Depende de cuál sea el que elija el planificador a corto plazo. Generalmente, esto se hace mediante un algoritmo de turno rotatorio (*round-robin*), cediendo el tiempo a cada proceso por turnos. También se pueden usar niveles de prioridad. Finalmente, hay una **cola de E/S** para cada dispositivo de E/S. Más de un proceso puede solicitar el uso del mismo dispositivo de E/S. Todos los procesos que esperan para utilizar cada dispositivo se introducen en la cola de ese dispositivo.

La Figura 8.11 sugiere la forma en que los procesos avanzan en el computador bajo el control del sistema operativo. Cada solicitud de proceso (desde los trabajos en cola, o los trabajos interactivos) se sitúa en la cola a largo plazo. A medida que los recursos están disponibles, una solicitud de proceso se hace proceso y pasa al estado preparado situándose en la cola de corto plazo. Alternativamente, el procesador ejecuta instrucciones del sistema operativo y de los procesos de usuario. Mientras el sistema operativo dispone del control, decide qué proceso de la cola de corto plazo debería ejecutarse a continuación. Cuando el sistema operativo ha terminado sus tareas inmediatas, devuelve el procesador a los procesos elegidos.

Como se mencionó anteriormente, un proceso en ejecución puede suspenderse por varias razones. Si se suspende porque el proceso solicita una E/S, se sitúa en la cola de E/S apropiada. Si se suspende porque ha transcurrido el tiempo que se le asignó o porque el sistema operativo debe atender alguna tarea urgente, se pone en estado preparado y se devuelve a la cola a corto plazo.

Finalmente, mencionaremos que el sistema operativo también gestiona las colas de E/S. Cuando finaliza una operación de E/S, el sistema operativo suprime de la cola de E/S el proceso atendido y lo sitúa en la cola de corto plazo. Después selecciona otro proceso en estado de espera (si lo hay) y actúa sobre el dispositivo de E/S correspondiente para que satisfaga la solicitud del proceso.

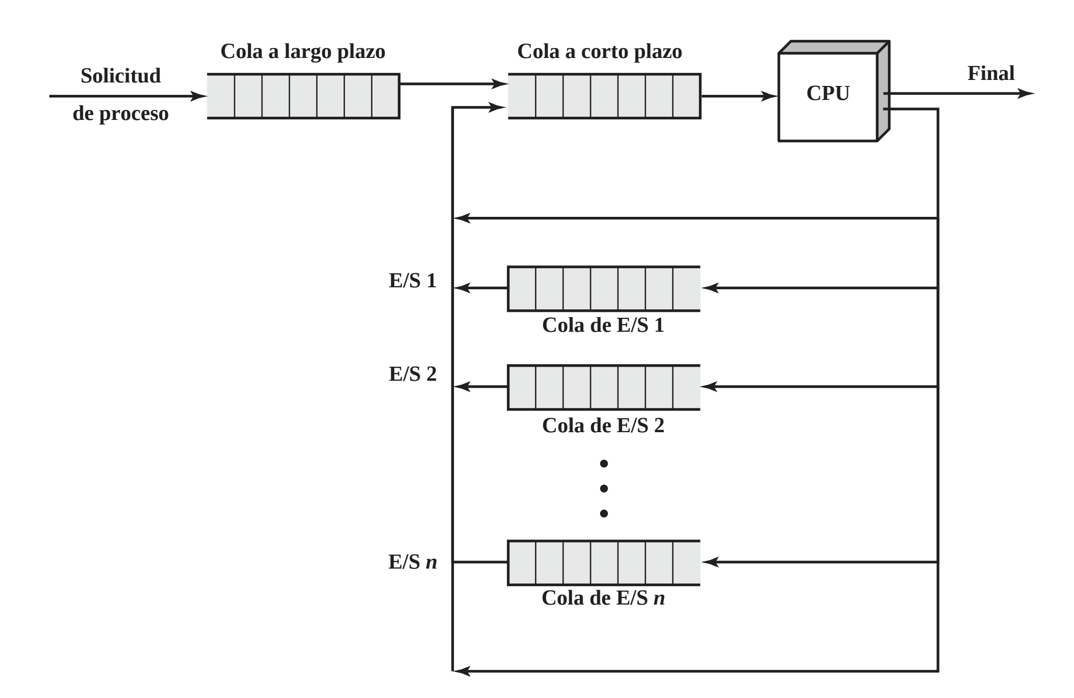


Figura 8.11. Representación de un diagrama de colas para la planificación del procesador.

8.3. GESTIÓN DE LA MEMORIA

En un sistema de monoprogramación, la memoria principal se divide en dos partes: una parte para el sistema operativo (el monitor residente) y otra parte para el programa que se está ejecutando. En un sistema multiprogramado, la parte de «usuario» de la memoria además debe subdividirse para dar cabida a los distintos procesos. La tarea de subdivisión la realiza dinámicamente el sistema operativo y se conoce como **gestión de memoria**.

Una gestión eficiente de la memoria es vital en un sistema multiprogramado. Si hay pocos procesos en memoria, puede ocurrir que todos los procesos estén esperando completar una E/S y el procesador permanecerá inactivo. En consecuencia, la memoria debe asignarse eficientemente para situar en memoria tantos procesos como sea posible.

INTERCAMBIO (*SWAPPING*)

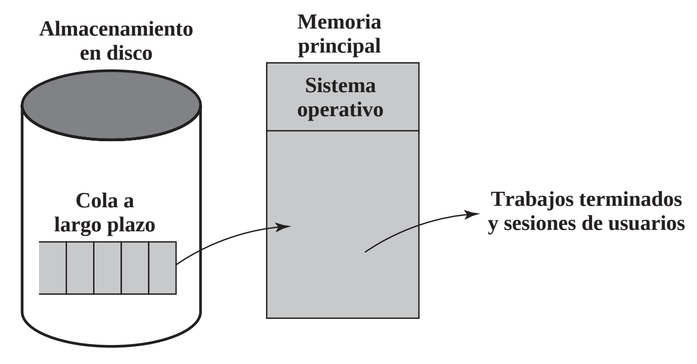
Volviendo a la Figura 8.11, se han discutido tres tipos de colas: la cola a largo plazo para solicitar procesos nuevos, la cola a corto plazo con los procesos preparados para utilizar el procesador y las distintas colas de E/S de los procesos que no están preparados para usar el procesador. Recuérdese que la razón última de estos mecanismos era que las actividades de E/S son mucho más lentas que el cálculo y que por consiguiente el procesador en un sistema con uniprogramación está la mayor parte del tiempo parado.

Sin embargo, el esquema de la Figura 8.11 no resuelve el problema por completo. Es verdad que, en este caso, la memoria contiene múltiples procesos y que el procesador puede conmutar a otro proceso cuando el proceso en curso tenga que esperar. Pero el procesador es tan rápido en comparación con las E/S que puede ser frecuente que *todos* los procesos de la memoria estén esperando una E/S. Por eso, incluso con la multiprogramación, un procesador puede estar parado la mayor parte del tiempo.

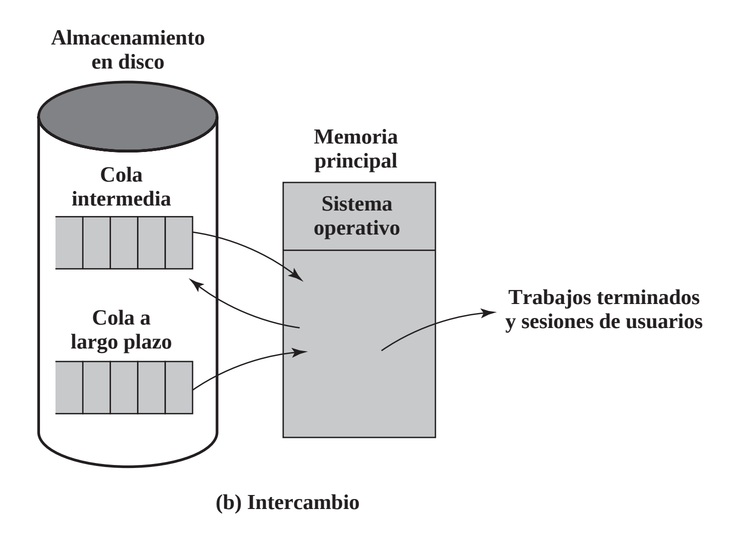
¿Qué se puede hacer? La memoria principal podría ampliarse y así ser capaz de dar cabida a más procesos. Pero hay dos problemas en esta solución. Primero, incluso hoy día la memoria principal es cara. Segundo, la necesidad de memoria de los programas ha crecido tan rápido como ha caido el costo de la memoria. Por eso una memoria mayor origina procesos mayores, no más procesos.

Otra solución es el *intercambio (swapping)*, representado en la Figura 8.12. Tenemos una cola a largo plazo de solicitudes de proceso, usualmente almacenado en disco. Estas solicitudes se traen a memoria, una a una, a medida que hay espacio disponible. Conforme terminan, los procesos se sacan de la memoria principal. Ahora, podría ocurrir que ninguno de los procesos en la memoria principal esté en el estado preparado (por ejemplo, todos están esperando una operación de E/S). En lugar de permanecer parado, el procesador *intercambia* uno de esos procesos situándolo en el disco en una *cola intermedia*. Esta es una cola de procesos existentes que se han sacado temporalmente de memoria. El sistema operativo entonces trae otro proceso de la cola intermedia, o acepta una nueva petición de proceso de la cola de largo plazo. La ejecución continúa con el proceso recientemente activado.

El intercambio es de hecho una operación de E/S, y por consiguiente existe la posibilidad de empeorar el problema más que de solucionarlo. No obstante, puesto que la E/S en disco es generalmente la operación de E/S más rápida (comparada con la E/S en cinta o mediante impresora),



(a) Planificación de trabajos simple



(b) Intercambio

Figura 8.12. Uso del intercambio (*swapping*).

usualmente el intercambio mejora las prestaciones. Un esquema más sofisticado, que implica el uso de la memoria virtual, mejora las prestaciones con respecto al intercambio simple. Esto se discutirá en breve. Pero primero, debemos proporcionar los fundamentos explicando la definición de particiones y la paginación.

DEFINICIÓN DE PARTICIONES

El esquema más simple para definir particiones en la memoria disponible es utilizar *particiones de tamaño fijo*, como muestra la Figura 8.13. Observe que, aunque las particiones son de tamaño fijo, no todas tienen igual tamaño. Cuando un proceso se introduce en memoria, se sitúa en la partición disponible más pequeña que puede incluirlo.

Incluso con el uso de particiones de distintos tamaños, se desperdiciará memoria. En la mayoría de los casos, un proceso no necesitará exactamente la memoria que proporciona una partición. Por ejemplo, un proceso que precise 3 MB de memoria se podría situar en la partición de 4 M de la Figura 8.13b, desperdiciando 1 M que podrían utilizarse para otro proceso.

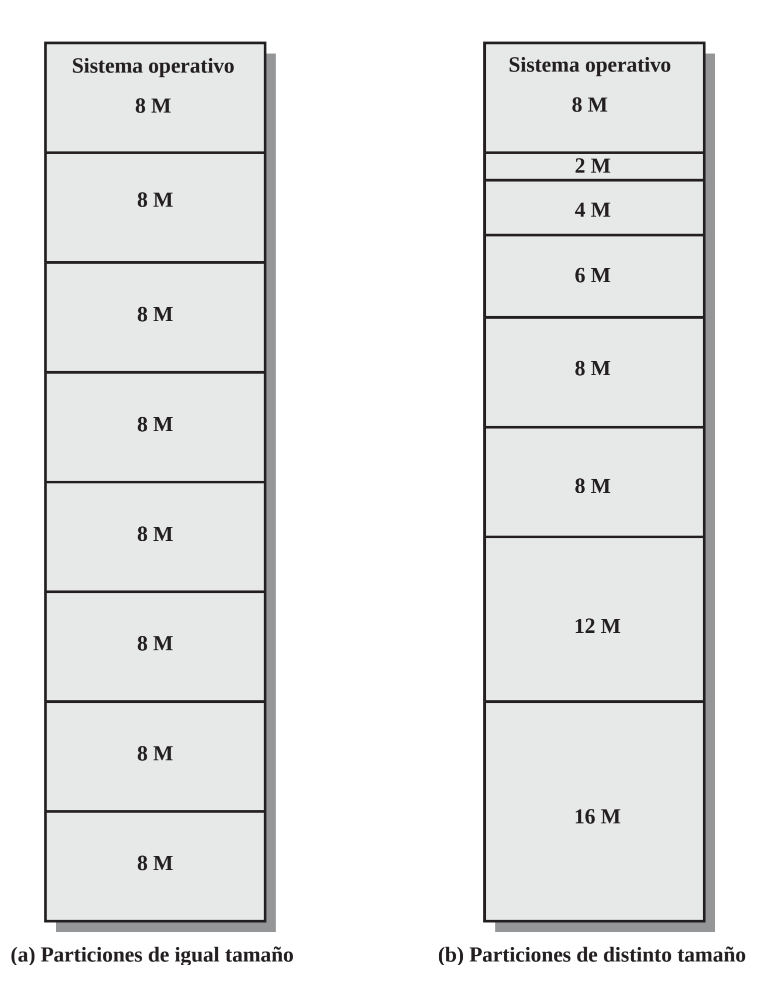


Figura 8.13. Ejemplo de particiones fijas de una memoria de 64 MB.

Una posibilidad más eficiente consiste en utilizar *particiones de tamaño variable*. Cuando un proceso se introduce en memoria, se le asigna exactamente la memoria que necesita y no más.

Ejemplo 8.2. En la Figura 8.14 se muestra un ejemplo utilizando 64 MB de memoria principal. Inicialmente, salvo por el sistema operativo, la memoria principal está vacía (a). Los primeros tres procesos se cargan, empezando por donde termina el sistema operativo y ocupando justo el espacio necesario para cada proceso (b,c,d). Esto deja un «hueco» al final de la memoria que es demasiado pequeño para un cuarto proceso. En cierto instante, ninguno de los procesos de memoria está preparado. El sistema operativo saca de memoria al proceso 2 (e), dejando espacio suficiente para cargar un nuevo proceso, el proceso 4 (f). Puesto que el proceso 4 es más pequeño que el proceso 2, se crea otro hueco pequeño. Posteriormente, se produce la situación en la que ninguno de los procesos que están en la memoria está preparado, excepto el proceso 2, que está disponible puesto que se encuentra en el estado Preparado-Suspendido. Como hay un espacio de memoria insuficiente para el proceso 2, el sistema operativo retira de memoria al proceso 1 (g) y vuelve a introducir al proceso 2 (h).

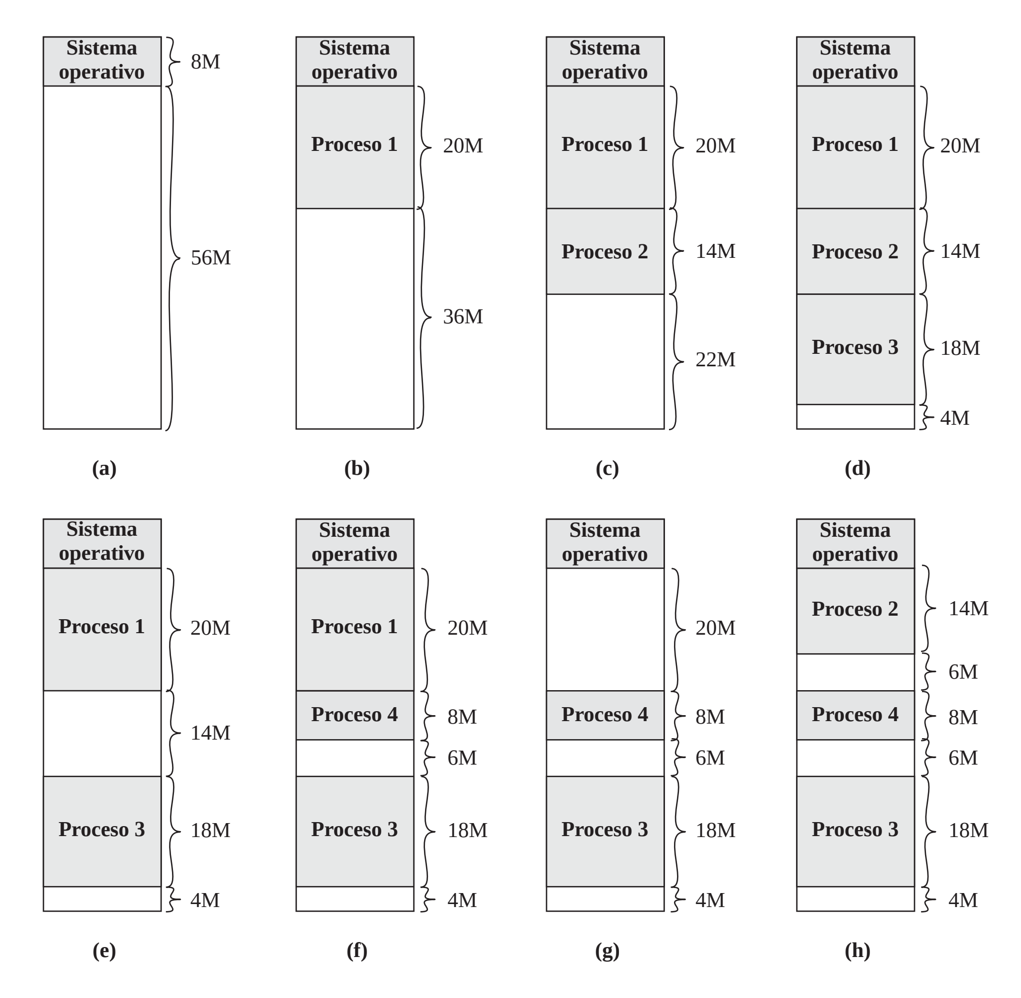


Figura 8.14. Efecto de la partición dinámica.

Como muestra este ejemplo, el método empieza bien, pero eventualmente puede llevar a situaciones en las que hay muchos huecos pequeños en memoria. A medida que pasa el tiempo, la memoria se fragmenta más y más, y empeora su utilización. Una técnica para solucionar este problema es la **compactación:** de vez en cuando, el sistema operativo desplaza los procesos en memoria para juntar toda la memoria libre en un bloque. Este es un procedimiento que consume parte del tiempo del procesador.

Antes de considerar formas de solucionar los problemas de la definición de particiones, debemos aclarar cierto extremo. Si el lector presta atención a la Figura 8.14 por un momento, resulta obvio que un proceso difícilmente se cargará en el mismo lugar de la memoria principal cada vez que se intercambia. Es más, si se utiliza compactación, un proceso puede desplazarse mientras se encuentra en memoria principal. La memoria del proceso está constituida por instrucciones y datos. Las instrucciones contendrán direcciones de posiciones de memoria de dos tipos:

- Direcciones de datos.
- Direcciones de instrucciones, usadas por las instrucciones de salto.

Pero estas direcciones no son fijas. Cambiarán cada vez que el proceso se intercambie. Para resolver este problema, se distingue entre direcciones lógicas y direcciones físicas. Una **dirección lógica** indica una posición relativa al comienzo del programa. Las instrucciones del programa contienen solo direcciones lógicas. Una **dirección física** es, por supuesto, la posición actual en la memoria principal. Cuando el procesador ejecuta un proceso, automáticamente convierte las direcciones lógicas en físicas sumando a cada dirección lógica la posición de comienzo actual del proceso, llamada **dirección base**. Este es otro ejemplo de un elemento hardware de la CPU diseñado para satisfacer las necesidades del sistema operativo. Las características exactas de este hardware dependen de la estrategia de gestión de memoria utilizada. Más adelante, en este mismo capítulo, veremos varios ejemplos.

PAGINACIÓN

Tanto las particiones de tamaño fijo como las de tamaño variable son ineficaces en el aprovechamiento de la memoria. Supóngase, no obstante, que la memoria se divide en trozos iguales de tamaño fijo y relativamente pequeño, y que cada proceso también se divide en pequeños trozos de tamaño fijo. Después los trozos de un programa, conocidos como **páginas**, se podrían asignar a los trozos de memoria disponibles, conocidos como **marcos (*frames*)**, o marcos de página. Entonces, el espacio de memoria desperdiciado por un proceso es, como mucho, una fracción de la última página.

La Figura 8.15 muestra un ejemplo del uso de las páginas y los marcos. En un instante dado, algunos de los marcos de memoria están ocupados y otros están libres. La lista de marcos libres es

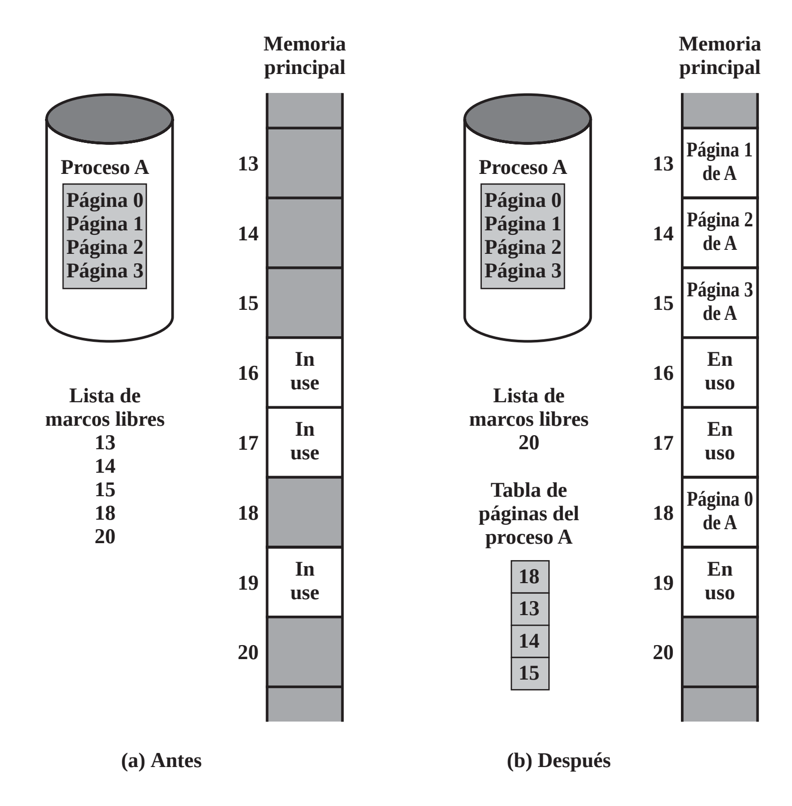


Figura 8.15. Asignación de marcos libres.

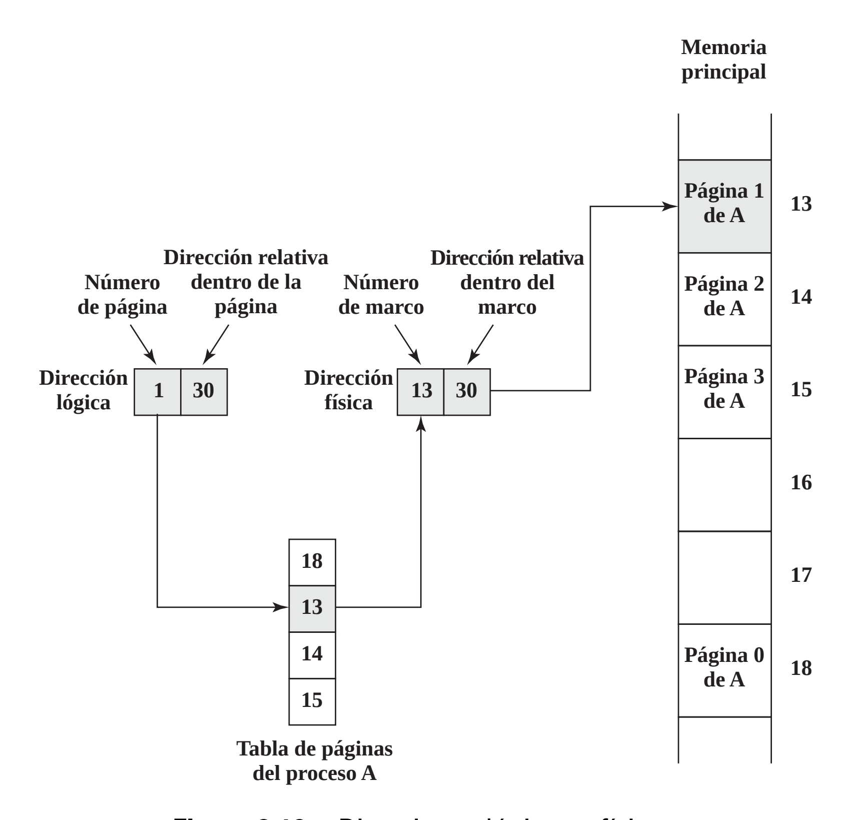


Figura 8.16. Direcciones lógicas y físicas.

gestionada por el sistema operativo. El Proceso A, almacenado en disco, consta de cuatro páginas. Cuando llega el momento de cargar este proceso, el sistema operativo encuentra cuatro marcos libres y carga las cuatro páginas del proceso A en cuatro marcos.

Supóngase ahora, como en este ejemplo, que no hay suficientes marcos contiguos sin utilizar para el proceso. (Hará esto que el sistema operativo no cargue A? La respuesta es no, porque de nuevo se utiliza una vez más el concepto de dirección lógica. Ya no es suficiente una única dirección de base. En cambio, el sistema operativo mantiene una **tabla de páginas** para cada proceso. La tabla de páginas indica el marco que aloja a cada página del proceso. Dentro del programa, cada dirección lógica está constituida por un número de página y una dirección relativa dentro de la página. Recuérdese que en el caso de particiones simples, una dirección lógica era la posición de una palabra en relación con el comienzo del programa; el procesador la traduce a una dirección física. Con la paginación, la traducción de dirección lógica a dirección física también la realiza el hardware del procesador. El procesador debe saber cómo acceder a la tabla de páginas del proceso en curso. A partir de la dirección lógica (número de página, dirección relativa), el procesador utiliza la tabla de páginas para generar la dirección física (número de marco, dirección relativa). Un ejemplo se muestra en la Figura 8.16.

Esta aproximación resuelve el problema anteriormente indicado. La memoria principal se divide en muchos marcos pequeños de igual tamaño. Cada proceso se divide en páginas del tamaño de los marcos: los procesos más pequeños necesitan menos páginas, los procesos mayores necesitan más. Cuando un proceso se lleva a memoria, sus páginas se cargan en los marcos disponibles y la tabla de páginas se actualiza.

MEMORIA VIRTUAL

Paginación por demanda. Con el uso de la paginación, se dispone de sistemas con multiprogramación verdaderamente efectivos. Es más, la sencilla táctica de dividir el proceso en páginas llevó al desarrollo de otro concepto decisivo: la memoria virtual.

Para entender la memoria virtual, debemos añadir una mejora al esquema de paginación discutido. Esta mejora es la **paginación por demanda**, que simplemente significa que cada página de un proceso se introduce en memoria solo cuando se necesita (es decir, cuando se solicita o demanda).

Considérese un proceso de tamaño elevado, consistente en un programa largo más un cierto número de matrices de datos. En un intervalo de tiempo corto, la ejecución puede confinarse a una pequeña sección del programa (por ejemplo una subrutina), y quizá solo se esté usando una o dos matrices de datos. Este es el principio de localidad, que se introdujo en el Apéndice 4A. Sería claramente un derroche cargar todas las páginas del proceso cuando solo se utilizarán unas pocas antes de que el proceso se suspenda. Podemos hacer un mejor uso de la memoria cargando solo unas pocas páginas. Después, si el programa salta a una instrucción de una página que no está en memoria principal, o si el programa hace referencia a un dato de una página que no está en memoria, se produce un **fallo de página**. Esto indica al sistema operativo que debe cargar la página deseada.

Así, en un instante dado, solo unas pocas páginas de un proceso están en memoria, y en consecuencia se pueden mantener en memoria más procesos. Además, se ahorra tiempo puesto que las páginas que no se utilizan no tienen que sufrir intercambios de almacenamiento. No obstante, el sistema operativo debe ser lo suficientemente ingenioso para manejar este esquema. Cuando se introduce una página en memoria, debe sacar otra fuera. Si saca una página justo en el momento en que va a empezar a utilizarse, tendrá que volver a introducirla en memoria casi inmediatamente; esto se denomina **reemplazo de página**. Si esto ocurre frecuentemente se produce una situación conocida como **hiperpaginación (*thrasing*)**: el procesador pasa la mayor parte de su tiempo intercambiando páginas en lugar de ejecutar instrucciones. Las formas de evitar la hiperpaginación constituyeron una importante área de investigación en los setenta que dio lugar a una variedad de algoritmos complejos pero efectivos. En esencia, el sistema operativo intenta predecir, basándose en su historia reciente, qué páginas se utilizarán con menos probabilidad en el futuro próximo.

Con la paginación por demanda, no es necesario cargar el proceso entero en la memoria principal. Este hecho tiene una consecuencia importante: es *posible que un proceso sea mayor que toda la memoria principal*. Una de las restricciones más importantes de la programación ha sido vencida. Sin paginación por demanda, un programador debe tener en cuenta la memoria disponible. Si el programa que se está escribiendo es demasiado largo, el programador debe buscar formas de estructurar el programa en trozos que puedan cargarse uno a uno. Con demanda de página, ese trabajo se deja al sistema operativo y al hardware. En lo que al programador concierne, él o ella disponen de una cantidad de memoria enorme, el tamaño asociado al espacio en disco.

Puesto que un proceso se ejecuta solo si está en memoria principal, esta recibe el nombre de **memoria real**. Pero el programador o usuario percibe una memoria mucho mayor —la que hay disponible en disco. En consecuencia, esta última se denomina **memoria virtual**. La memoria virtual posibilita una multiprogramación muy efectiva y libera al usuario de las innecesarias y exigentes restricciones de memoria principal.

Estructura de la tabla de páginas. El mecanismo básico para leer una palabra de memoria implica la traducción, mediante una tabla de páginas, de una dirección virtual o lógica, consistente en un número de página y un desplazamiento a una dirección física, constituida por un número de marco y un desplazamiento. Puesto que la tabla de páginas tiene una longitud variable, dependiendo del tamaño del proceso, no es posible almacenarla en los registros. En su lugar, debe accederse a ella en memoria principal. La Figura 8.16 sugiere una implementación hardware de este esquema. Cuando un proceso determinado está ejecutándose, un registro contiene la dirección de inicio de la tabla de páginas de ese proceso. El número de página de una dirección virtual se utiliza como índice en la tabla para buscar el correspondiente número de marco. Este se combina con la parte de desplazamiento de la dirección virtual para construir la dirección real deseada.

En la mayoría de los sistemas, hay una tabla de páginas por proceso. Pero cada proceso puede ocupar una gran cantidad de memoria virtual. Por ejemplo, en la arquitectura VAX, cada proceso puede tener hasta $2^{31} = 2$ GB de memoria virtual. Utilizando páginas de $2^9 = 512$ bytes, eso significa que se necesitan tablas de páginas de 2^{22} elementos *por proceso*. Claramente, la cantidad de memoria dedicada solo a tablas de páginas podría ser inaceptablemente alta. Para solucionar este problema, la mayoría de los esquemas de memoria virtual almacenan las tablas de páginas en la memoria virtual en lugar de en la memoria real. Esto significa que la tabla de páginas también está sujeta a paginación igual que el resto de páginas. Cuando un proceso se está ejecutando, al menos una parte de su tabla de páginas, incluyendo el elemento correspondiente a la página actualmente en ejecución, debe estar en la memoria principal. Algunos procesadores hacen uso de un esquema de dos niveles para organizar las tablas de páginas grandes. En este esquema, hay una página de directorio en la que cada elemento apunta a una tabla de páginas. Así, si la longitud de la página de directorio es X, y si la longitud máxima de una tabla de páginas es Y, un proceso puede estar constituido por hasta $X \times Y$ páginas. Típicamente, la longitud máxima de una tabla de páginas se restringe al tamaño de una página. Veremos un ejemplo de esta aproximación de dos niveles más adelante en este mismo capítulo, cuando estudiemos el Pentium II.

Una aproximación alternativa al uso de tablas de páginas de uno o dos niveles es el uso de una estructura de *tabla de páginas invertida* (Figura 8.17). Variaciones de esta aproximación se utilizan en el PowerPC, el UltraSPARC y la arquitectura IA-64. Una implementación del sistema operativo Mach en el RT-PC también usa esta técnica.

En esta aproximación, la porción de la dirección virtual correspondiente al número de página se mapea en una tabla de dispersión (*hash*) mediante una función de dispersión sencilla[2]. La tabla de dispersión incluye un puntero a una tabla de páginas invertida, que contiene los elementos de la tabla de páginas. Con esta estructura, hay un elemento en la tabla de dispersión y en la tabla de páginas invertida para cada página de memoria real en vez de para cada página de memoria virtual. Así, se necesita una porción fija de la memoria real para las tablas independientemente del número de procesos o páginas virtuales que se admitan. Puesto que más de una dirección virtual puede apuntar al mismo elemento de la página de dispersión, se utiliza una técnica de encadenamiento para solucionar

---

[2] Una función de dispersión asocia números comprendidos entre 0 y M con números entre 0 y N, donde $M > N$. La salida de la función de dispersión se utiliza como índice en la tabla de dispersión. Puesto que más de una entrada se asocia a la misma salida, es posible que un elemento de entrada apunte a una posición de la tabla de dispersión que está ocupada. En ese caso, el nuevo elemento debe pasar a otra posición de la tabla de mezcla. Usualmente, el nuevo elemento se sitúa en la primera posición vacía que se encuentra, y se establece un puntero desde la posición original para encadenar juntas las posiciones que se van ocupando. *Véase* [STAL98] para una discusión más detallada de las tablas de dispersión.

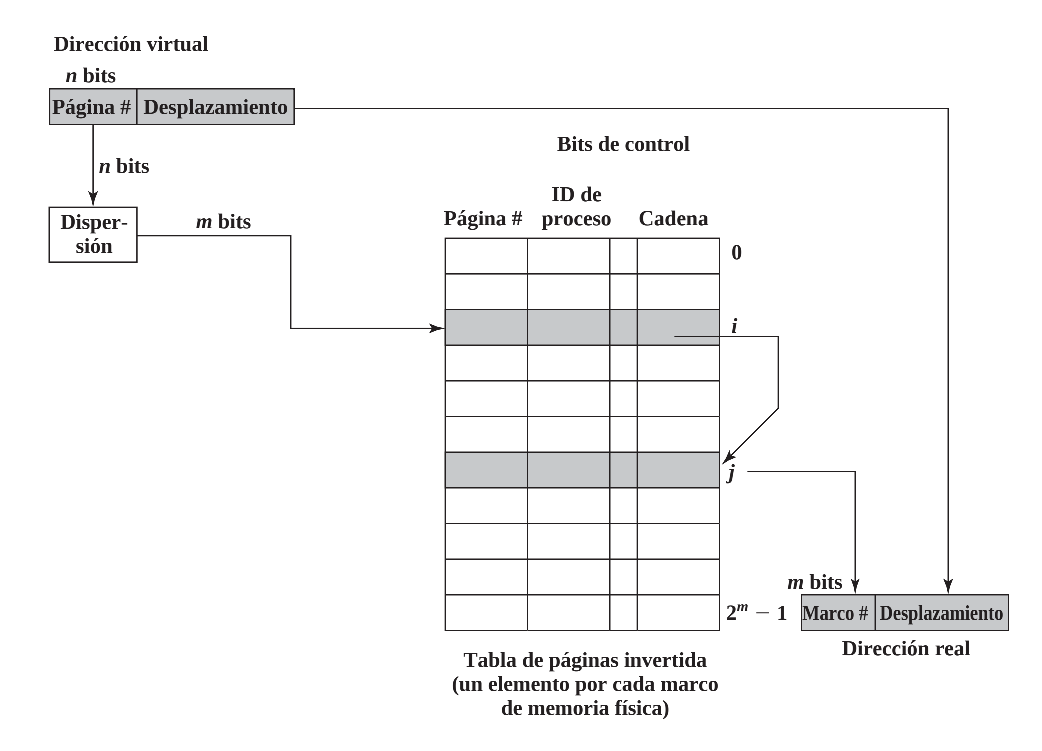


Figura 8.17. Estructura de la tabla de páginas invertida.

este problema. La técnica de dispersión da lugar a cadenas usualmente cortas —con uno o dos elementos. La estructura de la tabla de páginas se denomina estructura invertida debido a que indexa elementos de la tabla de páginas a través del número de marco en lugar de mediante el número de página virtual.

BUFFER DE TRADUCCIÓN ANTICIPADA (*TRANSLATION LOOKASIDE BUFFER*)

En principio, toda referencia a memoria virtual puede ocasionar dos accesos a la memoria física: uno para captar el elemento de la tabla de páginas apropiada, y otro para captar el dato deseado. Como consecuencia, un esquema de memoria virtual directo tendría el efecto de duplicar el tiempo de acceso a memoria. Para resolver este problema, la mayoría de los esquemas de memoria virtual hacen uso de una caché especial para los elementos de la tabla de páginas, llamada usualmente buffer de traducción anticipada (TLB, *Translation Lookaside Buffer*). Este buffer funciona de la misma manera que una memoria caché y contiene aquellos elementos de la tabla de páginas a los que se ha accedido más recientemente. La Figura 8.18 es un diagrama de flujo que muestra el uso del TLB. Por el principio de localidad, la mayoría de las referencias a memoria corresponderán a posiciones de las páginas recientemente usadas. Por eso, la mayoría de las referencias implican a elementos de la tabla de páginas incluidas en el TLB. Estudios del TLB de VAX muestran que este esquema puede mejorar significativamente las prestaciones [CLAR85, SATY81].

Observe que el mecanismo de memoria virtual debe interactuar con el sistema de caché (no con la caché que implementa el TLB, sino con la caché de la memoria principal). Esto se ilustra en la

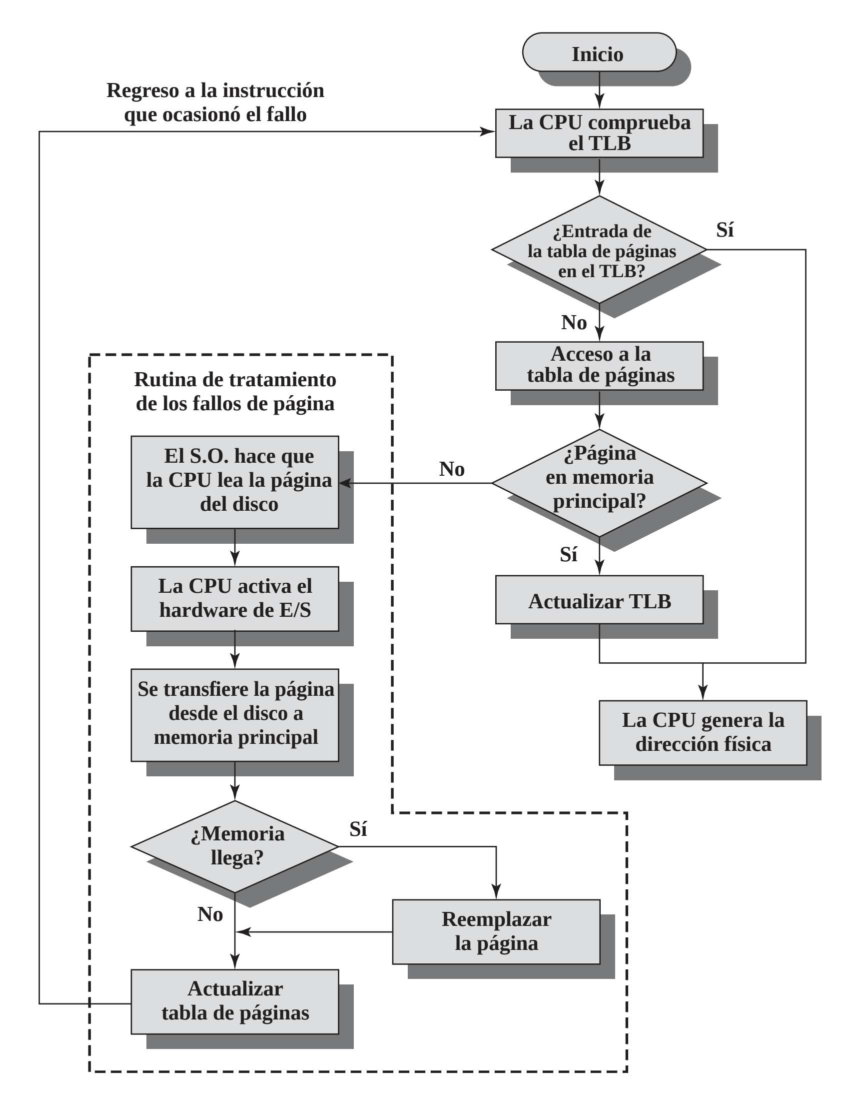


Figura 8.18. Funcionamiento de la paginación y del buffer de traducción anticipada (TLB) [FURH87].

Figura 8.19. Una dirección virtual estará generalmente en la forma de número de página más desplazamiento. Primero, el sistema de memoria consulta el TLB para comprobar si hay coincidencia con algún elemento de la tabla de páginas incluido en él. Si es así, se genera la dirección real (física) combinando el número de marco con el desplazamiento. Si no, se accede al elemento correspondiente de la tabla de páginas. Una vez que se ha generado la dirección real, constituida por una marca y los bits restantes (*véase* la Figura 4.17), se consulta la caché para ver si el bloque que contiene la palabra está presente. Si es así, se envía al procesador. Si no, se busca la palabra en memoria principal.

El lector puede apreciar la complejidad del hardware del procesador implicado en una simple referencia a memoria. La dirección virtual es traducida a una dirección real. Esto implica una referencia a la tabla de páginas, que puede estar en el TLB, en memoria principal o en disco. La palabra referenciada puede estar en caché, en memoria principal o en disco. En este último caso, la página

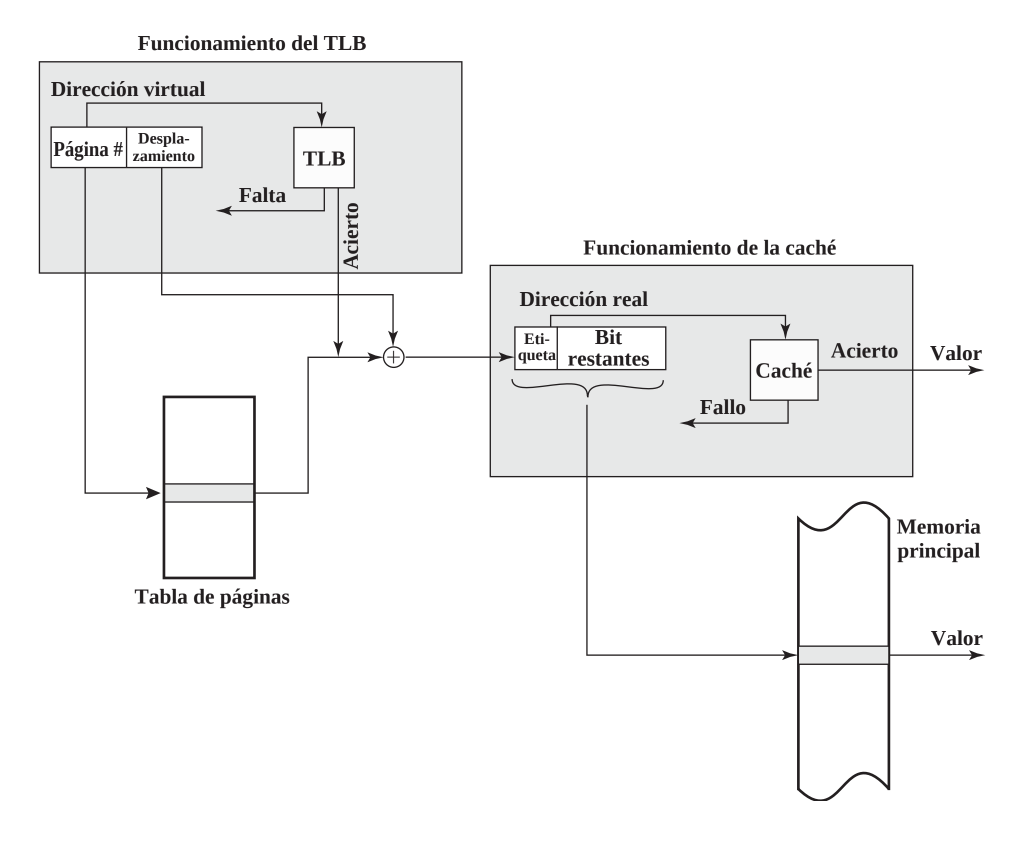


Figura 8.19. Buffer de traducción anticipada y funcionamiento de la caché.

que contiene a la palabra debe cargarse en la memoria principal y su bloque pasar a la caché. Además, el elemento de la tabla de páginas correspondiente a esa página debe actualizarse.

SEGMENTACIÓN

Hay otra forma en la que puede subdividirse la memoria direccionable, conocida como *segmentación*. Mientras que la paginación es invisible para el programador y sirve para proporcionar al programador un espacio de direcciones mayor, la segmentación es usualmente visible para el programador y proporciona una forma conveniente de organizar los programas y los datos, para asociar los privilegios y los atributos de protección con las instrucciones y los datos.

La segmentación permite que el programador vea la memoria constituida por múltiples espacios de direcciones o segmentos. Los segmentos tienen un tamaño variable, dinámico. Usualmente, el programador o el sistema operativo asignarán programas y datos a segmentos distintos. Puede haber segmentos de programa distintos para varios tipos de programas y también distintos segmentos de datos. Se pueden asignar a cada segmento derechos de acceso y uso. Las referencias a memoria se realizan mediante direcciones constituidas por un número de segmento y un desplazamiento.

Esta organización tiene ciertas ventajas para el programador frente a un espacio de direcciones no segmentado:

1. Simplifica la gestión de estructuras crecientes de datos. Si el programador no conoce a priori el tamaño que puede llegar a tener una estructura de datos particular, no es necesario que lo presuponga. A la estructura de datos se le asigna su propio segmento y el sistema operativo lo expandirá o lo reducirá según sea necesario.
2. Permite modificar los programas y recompilarlos independientemente, sin que sea necesario volver a enlazar y cargar el conjunto entero de programas. De nuevo, esto se consigue utilizando varios segmentos.
3. Permite que varios procesos compartan segmentos. Un programador puede situar un programa correspondiente a una utilidad o una tabla de datos de interés en un segmento que puede ser direccionado por otros procesos.
4. Se facilita la protección. Puesto que un segmento se construye para contener un conjunto de programas o datos bien definido, el programador o el administrador del sistema puede asignar privilegios de acceso de forma adecuada.

Estas ventajas no se tienen con la paginación, que es invisible para el programador. Por otra parte, hemos visto que la paginación proporciona una forma eficiente de gestionar la memoria. Para combinar las ventajas de ambas, algunos sistemas están equipados con el hardware y el software del sistema operativo que permite las dos.

8.4. GESTIÓN DE MEMORIA EN EL PENTIUM II Y EN EL POWERPC

HARDWARE DE GESTIÓN DE MEMORIA EN EL PENTIUM II

Desde la introducción de las arquitecturas de 32 bits, los microprocesadores han desarrollado esquemas de gestión de memoria sofisticados aprovechando la experiencia obtenida con los sistemas de medio y gran tamaño. En muchos casos, las versiones de los microprocesadores son superiores a sistemas de media y gran escala que les antecedieron. Puesto que los sistemas fueron desarrollados por los fabricantes del hardware de microprocesadores y debieran poder utilizarse con una cierta variedad de sistemas operativos, tienden bastante a ser de uso general. Un ejemplo representativo es el esquema utilizado por el Pentium II. El hardware de gestión de memoria del Pentium II es esencialmente el mismo que se usa en los procesadores 80386 y 80486 de Intel, con ciertas mejoras.

Espacios de direcciones. El Pentium II incluye hardware tanto para segmentación como para paginación. Ambos mecanismos se pueden desactivar, permitiendo elegir entre cuatro formas de ver la memoria:

- **Memoria no segmentada y no paginada:** en este caso, la dirección virtual es la misma que la dirección física. Esto es útil, por ejemplo, cuando se utiliza como controlador de baja complejidad y elevadas prestaciones.
- **Memoria paginada no segmentada:** la memoria se ve como un espacio de direcciones lineal paginado. La protección y la gestión de memoria se realiza via paginación. Esta es la forma preferida por ciertos sistemas operativos (por ejemplo, el UNIX de Berkeley).

- **Memoria segmentada no paginada:** se ve la memoria como un conjunto de espacios de direcciones lógicas. La ventaja de esta imagen sobre el enfoque de la paginación estriba en que proporciona protección por debajo del nivel de byte, si es necesario. Es más, a diferencia de la paginación, garantiza que la tabla de traducción necesaria (la tabla de segmentos) se encuentra almacenada en el chip cuando el segmento está en memoria. De esta forma, la segmentación sin páginas da lugar a tiempos de acceso predecibles.
- **Memoria segmentada paginada:** se utiliza la segmentación para definir particiones lógicas de memoria en el control de acceso, y la paginación se usa para gestionar la asignación de memoria dentro de las particiones. Ciertos sistemas operativos tales como el UNIX System V prefieren esta visión de la memoria.

Segmentación. Cuando se utiliza segmentación, cada dirección virtual (llamada dirección lógica en la documentación del Pentium) consta de una referencia al segmento de 16 bits y un desplazamiento de 32 bits. Dos bits de la referencia al segmento se utilizan para el mecanismo de protección, y los 14 bits restantes para especificar al segmento en cuestión. Así, con una memoria no segmentada, la memoria virtual de usuario es $2^{32} = 4$ GB. Con una memoria segmentada, el espacio de memoria virtual total visto por el usuario es $2^{46} = 64$ terabytes (TB). El espacio de direcciones físicas emplea direcciones de 32 bits, con un máximo de 4 GB.

El volumen total de memoria virtual puede ser mayor de 64 TB. Esto se debe a que la forma de interpretar una dirección virtual por parte del procesador depende de la forma en que esté activo en un momento dado. Una mitad del espacio de direcciones virtuales (8K segmentos $\times$ 4 GBytes) es global, compartida por todos los procesos; es resto de la memoria es local y distinta para cada proceso.

Hay dos formas de protección asociadas a cada segmento: nivel de privilegio y atributo de acceso. Hay cuatro niveles de privilegio desde el más protegido (nivel 0) al menos protegido (nivel 3). El nivel de privilegio asociado a un segmento de datos es su «clasificación»; el nivel de privilegio asociado con un segmento de programa es su «acreditación» (*clearance*). Un programa en ejecución puede acceder a un segmento de datos solo si su nivel de acreditación es menor (mayor privilegio) o igual (igual privilegio) que el nivel de privilegio del segmento de datos.

El hardware no indica cómo deben utilizarse estos niveles de privilegio; esto depende del diseño y de la implementación del sistema operativo. El nivel de privilegio 1 sería utilizado por la mayor parte del sistema operativo, y el nivel 0 por una pequeña parte del mismo dedicada a la gestión de memoria, la protección y el control del acceso. Esto deja dos niveles para las aplicaciones. En muchos sistemas, las aplicaciones se encuentran en el nivel 3, dejándose sin utilizar el nivel 2. Los subsistemas de aplicación específica que deben protegerse debido a que implementan sus propios mecanismos de seguridad son buenos candidatos para situarse en el nivel 2. Algunos ejemplos son los sistemas de gestión de bases de datos, sistemas de automatización de oficinas y entornos de ingeniería del software.

Además de regular el acceso a los segmentos de datos, el mecanismo de privilegio limita el uso de ciertas instrucciones. Algunas instrucciones, tales como las que utilizan los registros de gestión de memoria, solo pueden ejecutarse en el nivel 0. Las instrucciones de E/S solo pueden ejecutarse en cierto nivel determinado por el sistema operativo; éste suele ser el nivel 1.

El atributo de acceso al segmento de datos especifica si se permiten accesos de lectura-escritura o solo de lectura. Para los segmentos de programa, el atributo de acceso especifica acceso de lectura-ejecución o de solo-lectura.

El mecanismo de traducción de dirección para la segmentación implica hacer corresponder una dirección virtual con lo que se denomina una dirección lineal (Figura 8.20b). Una dirección virtual consiste en un desplazamiento de 32 bits y un selector de segmento de 16 bits (Figura 8.20a). El selector de segmentos consta de los siguientes campos:

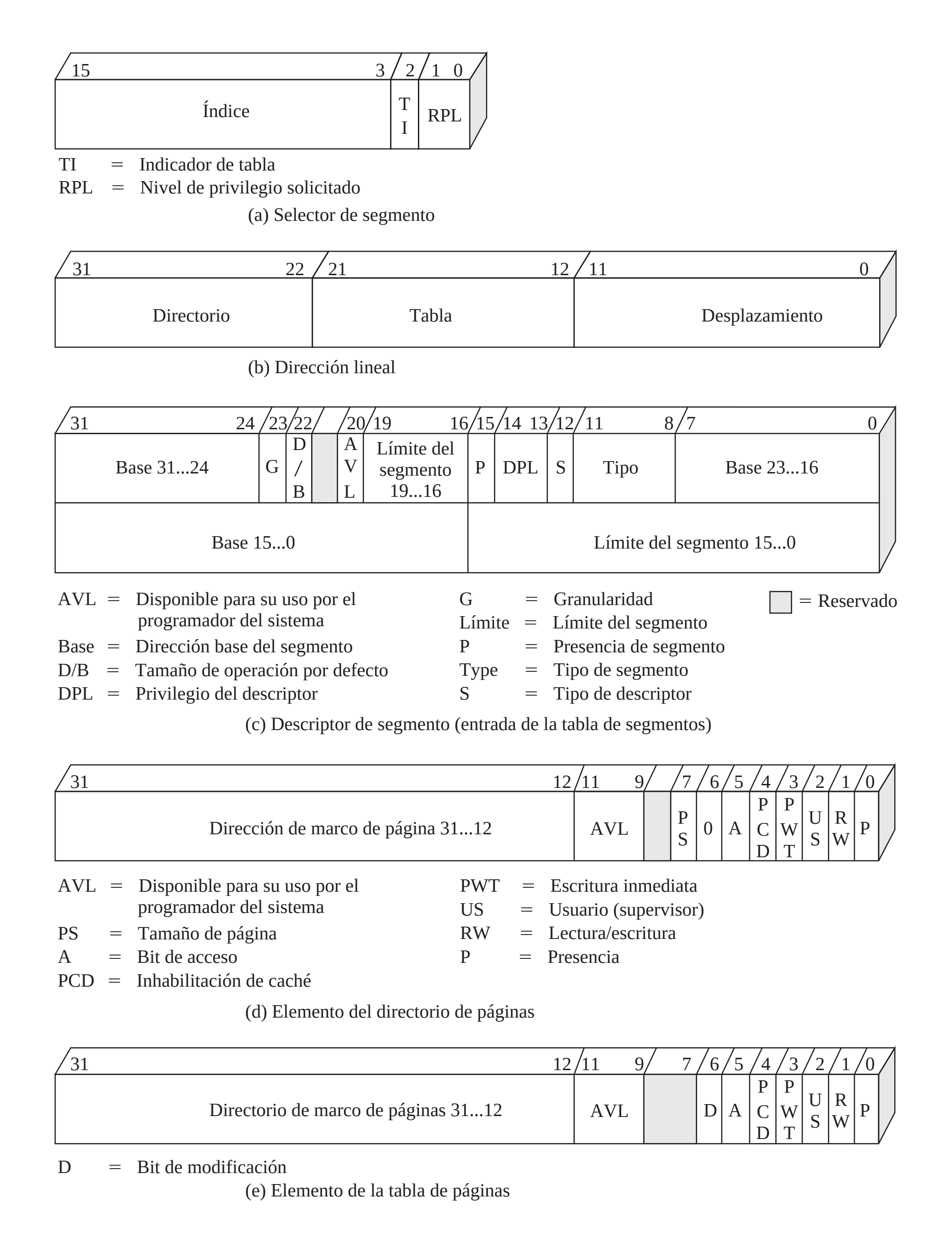


Figura 8.20. Formatos para la gestión de memoria en el Pentium II.

- **Indicador de tabla (TI, *Table Indicator*):** indica si para la traducción se va a utilizar la tabla de segmento global o la tabla de segmento local.
- **Número de segmento:** el número del segmento. Sirve como un índice en la tabla de segmentos.
- **Nivel de privilegio solicitado (RPL, *Requested Privilege Level*):** el nivel de privilegio para el acceso en cuestión.

Cada elemento en la tabla de segmentos consta de 64 bits, como muestra la Figura 8.20c. Los campos se definen en la Tabla 8.5.

Tabla 8.5. Parámetros para la gestión de memoria en el Pentium II.

Descriptor de segmentos (elemento en la tabla de segmentos)
Base Define la dirección de comienzo del segmento dentro del espacio lineal de direcciones de cuatro GB.
Bit D/B En un segmento de código, este es el bit D, e indica si los operandos y modos de direccionamiento son de 16 o 32 bits.
Nivel de privilegio del descriptor (DPL, *Descriptor Privilege Level*) Especifica el nivel de privilegio del segmento al que se refiere el descriptor de segmento en cuestión.
Bit de granularidad (G) Indica si el campo límite debe ser interpretado en unidades de un byte o de cuatro KB.
Límite Define el tamaño del segmento. El procesador interpreta el campo Límite de dos formas posibles según el bit de granularidad: en unidades de un byte, hasta un límite de un MB en el tamaño del segmento, o en unidades de cuatro KB, hasta un límite de cuatro GB en el tamaño del segmento.
Bit S Determina si un segmento dado es un segmento del sistema, o un segmento de código o de datos.
Bit de segmento presente (P) Lo usan los sistemas no paginados. Indica si el segmento está disponible en memoria principal. En los sistemas paginados este bit está siempre a 1.
Tipo Distingue entre varios tipos de segmentos e indica los atributos de acceso.
Elementos del directorio de páginas y de la tabla de páginas
Bit de acceso (A, *Accesed bit*) El procesador pone este bit a 1 en ambos niveles de las tablas de página cuando se produce una operación de lectura o escritura en la página correspondiente.

(Continúa)

Tabla 8.5. Parámetros para la gestión de memoria en el Pentium II *(continuación).*

Bit de modificación (D, *Dirty bit*)
El procesador pone a 1 este bit cuando se produce una operación de escritura en la página correspondiente.

Dirección del marco de página
Proporciona la dirección física de la página en la memoria si el bit de presencia está activo. Puesto que los marcos de página se alinean con los bloques de 4K, los doce bits inferiores son cero, y solo los veinte bits superiores se incluyen en el elemento. En el directorio de páginas, la dirección es la de la tabla de páginas.

Bit de inhabilitación de caché para la página (PCD, *Page Cache Disable bit*)
Indica si los datos de la página se pueden introducir en caché.

Bit de tamaño de página (PS)
Indica si la página es de cuatro KB o de cuatro MB.

Bit de escritura inmediata de página (PWT, *Page Write Through bit*)
Indica si se utiliza la política de escritura inmediata (*Write Through*) o la de post escritura (*Write Back*) para actualizar los datos en la página correspondiente.

Bit de presencia (P, *Present bit*)
Indica si la tabla de páginas o la página está en memoria principal.

Bit de lectura/escritura (RW, *Read/Write bit*)
En las páginas del nivel de usurario, indica si los programas de usuario pueden acceder a la página solo para lectura, o para lectura y escritura.

Bit de usuario/supervisor (US)
Indica si la página es accesible solo para el sistema operativo (nivel supervisor) o está disponible tanto para el sistema operativo como para las aplicaciones (nivel de usuario).

Paginación. La segmentación es una propiedad opcional y puede desactivarse. Cuando se utiliza la segmentación, las direcciones utilizadas en los programas son direcciones virtuales y se convierten en direcciones lineales, como se ha descrito. Cuando no se utiliza segmentación, los programas utilizan direcciones lineales. En cualquiera de los casos, el siguiente paso es la traducción de una dirección lineal a una dirección real de 32 bits.

Para comprender la estructura de la dirección lineal, es preciso tener en cuenta que el mecanismo de paginación del Pentium II es de hecho una operación de búsqueda en una tabla de dos niveles. El primer nivel es un directorio de páginas, que contiene hasta 1024 elementos. Esto divide los cuatro GB del espacio lineal de memoria en 1024 grupos de páginas, cada uno con su propia tabla de páginas, y cada uno de cuatro MB de longitud. La gestión de memoria permite la opción de utilizar un directorio de páginas para todos los procesos, un directorio de páginas para cada proceso o una combinación de los dos. El directorio de páginas para la tarea en curso está siempre en memoria principal. Las tablas de páginas pueden estar en memoria virtual.

La Figura 8.20 muestra los formatos de los elementos de los directorios de páginas y las tablas de páginas. Los campos se definen en la Tabla 8.5. Obsérvese que los mecanismos de control de acceso pueden proporcionarse en base a una página o a un grupo de páginas.

Además, el Pentium II hace uso del buffer de traducción rápida (TLB). El buffer puede contener 32 elementos de la tabla de páginas. Cada vez que cambia el directorio de páginas, el buffer se borra.

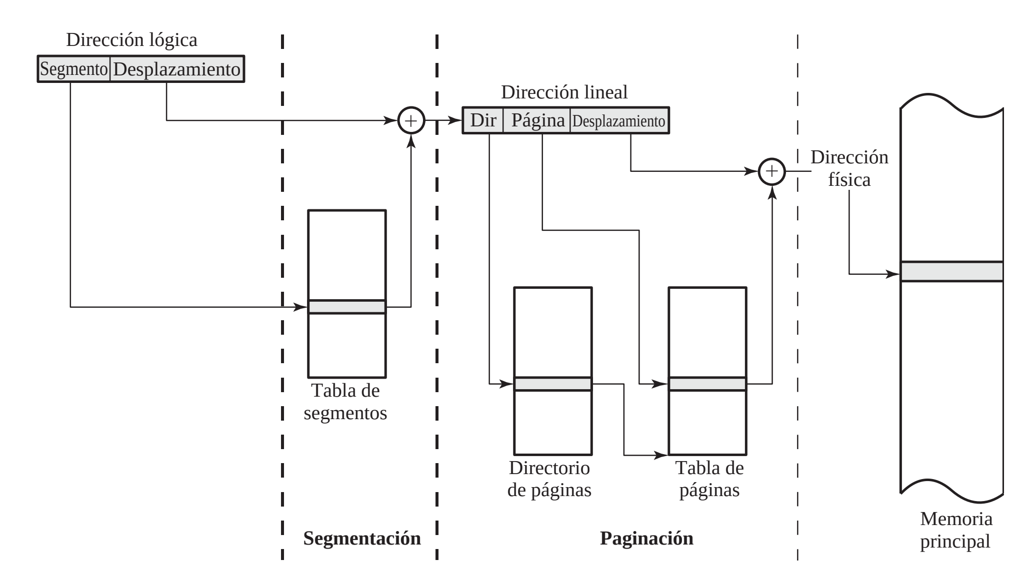


Figura 8.21. Mecanismos de traducción de una dirección de memoria en el Pentium II.

La Figura 8.21 ilustra la combinación de los mecanismos de segmentación y paginación. Por claridad, los mecanismos del TLB y de la memoria caché no se muestran.

Finalmente, el Pentium II incluye una nueva ampliación que no existe ni en el 80386 ni en el 80486, se permiten dos tamaños de páginas. Si el bit PSE (extensión de tamaño de página) del registro de control 4 está a 1, la unidad de paginación permite que el programador del sistema operativo defina la página con un tamaño de cuatro KB o de cuatro MB.

Cuando se utilizan páginas de cuatro MB, hay un solo nivel en la tabla de búsqueda de páginas. Cuando el hardware accede al directorio de páginas, el elemento del directorio de páginas (Figura 8.20d) tiene el bit PS a 1. En este caso, se ignoran desde el bit 9 al bit 21 y los bits desde 22 al 31 definen la dirección base de una página de memoria de cuatro MB. Así, hay una sola tabla de páginas.

El uso de páginas de cuatro MB reduce las necesidades de almacenamiento para la gestión de memoria en memorias principales grandes. Con páginas de cuatro KB, una memoria principal de cuatro GBytes necesita del orden de cuatro MB de memoria solo para la tabla de páginas. Con páginas de cuatro MB, una única tabla, de cuatro KB de longitud, es suficiente para la gestión las páginas.

HARDWARE DE GESTIÓN DE MEMORIA EN EL POWERPC

El PowerPC proporciona un amplio conjunto de mecanismos de direccionamiento. Para las implementaciones de la arquitectura de 32 bits, existe un esquema de paginación con un mecanismo sencillo de segmentación. Para las implementaciones de 64 bits, es posible la paginación y un mecanismo más potente de segmentación. Además, tanto para las máquinas de 32 como de 64 bits hay un mecanismo hardware alternativo, conocido como traducción de dirección de bloque. Brevemente, el esquema de direccionamiento de bloque está diseñado para resolver un problema de los mecanismos de paginación. Con la paginación, un programa puede hacer referencias frecuentes a un número elevado de páginas. Por ejemplo, los programas que utilizan tablas del sistema operativo o buffers para tramas

gráficas pueden tener este comportamiento. Como resultado, las páginas frecuentemente usadas pueden estar constantemente introduciéndose y sacándose de memoria. El direccionamiento de bloque permite al procesador definir cuatro bloques grandes de la memoria de instrucciones y cuatro bloques grandes de la memoria de datos sobre los que no se aplica el mecanismo de paginación.

Una discusión del direccionamiento de bloque está fuera del alcance de este capítulo. En esta subsección, nos concentramos en los mecanismos de paginación y segmentación del PowerPC de 32 bits. El esquema para el de 64 bits es similar.

El PowerPC de 32 bits utiliza direcciones efectivas de 32 bits (Figura 8.22a). La dirección incluye un identificador de página de 16 bits y un selector de byte de 12 bits. Así pues, se utilizan páginas de $2^{12} = 4$ KB. Son posibles hasta $2^{16} = 64$K páginas por segmento. Cuatro bits de la dirección se utilizan para designar uno de los 16 registros de segmento. Los contenidos de estos registros están controlados por el sistema operativo. Cada registro de segmento incluye bits de control de acceso y un identificador de 24 bits, de manera que una dirección efectiva de 32 bits se hace corresponder a una dirección virtual de 52 bits (Figura 8.23).

El PowerPC utiliza una sola tabla de páginas invertida. La dirección virtual se utiliza como un índice en la tabla de páginas que actúa de la siguiente manera. En primer lugar, se calcula un código de mezcla como sigue:

$$H(0\ldots 18) = SID(5\ldots 23) \oplus VPN(0\ldots 18)$$

El número de página virtual (VPN) en la dirección de página virtual se completa añadiendo a su izquierda (extremo más significativo) tres ceros, para formar un número de 19 bits. Entonces se realiza la operación *exclusive-or* bit a bit de este número con los 19 bits más a la derecha del identificador

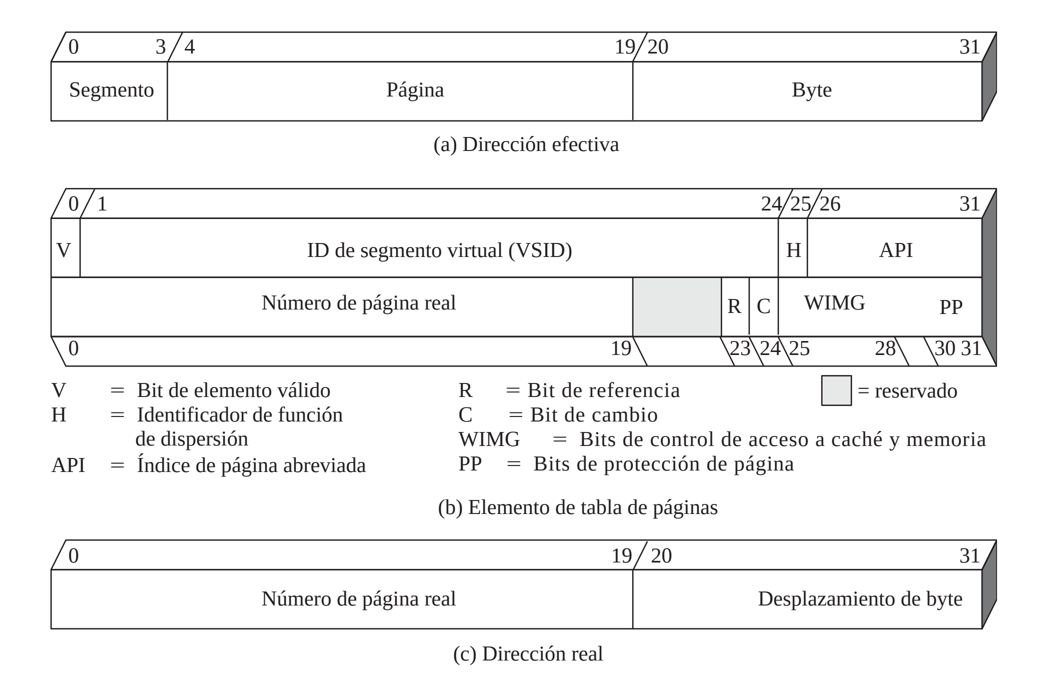


Figura 8.22. Formatos para la gestión de memoria en el PowerPC de 32 bits.

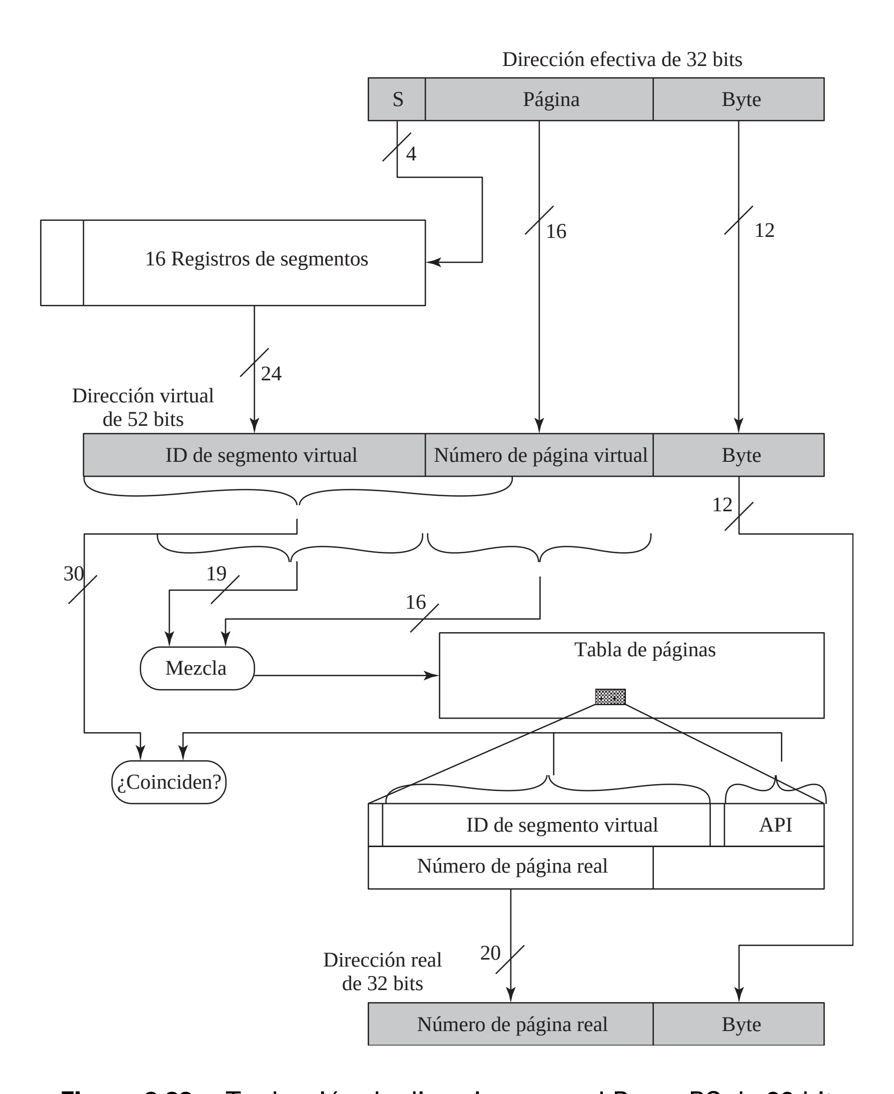


Figura 8.23. Traducción de direcciones en el PowerPC de 32 bits.

de segmento virtual (SID) para formar un código de mezcla (H) de 19 bits. La tabla está organizada en *n* grupos de ocho elementos. De 10 a 19 bits del código de mezcla (según el tamaño de la tabla de páginas) se utilizan para seleccionar uno de los grupos de la tabla. El hardware de gestión de memoria comprueba luego los ocho elementos del grupo para determinar si hay alguna coincidencia con la dirección virtual.

Para determinar si hay coincidencia, cada elemento de la tabla de páginas incluye un identificador (ID) de segmento virtual y los seis bits más a la izquierda del número de página virtual, llamado índice de página abreviado (puesto que al menos diez bits del número de página virtual siempre participan en la función de mezcla para seleccionar un grupo de elementos de la tabla de páginas, solo se necesita almacenar en el elemento de la tabla de páginas una forma abreviada del número de página virtual para comprobar la dirección virtual). Si hay coincidencia, entonces se obtiene de la dirección un número de veinte bits correspondiente a la página real, que se concatena con los doce bits menos significativos de la dirección efectiva para formar la dirección física de 32 bits a la que se accede.

Si no hay coincidencia, el código de mezcla se complementa para producir un nuevo índice de la tabla de páginas que está en la misma posición con repecto al extremo opuesto de la tabla. Este grupo se comprueba para determinar si hay coincidencia. Si no la hay, se produce una interrupción de falta de página.

La Figura 8.22 muestra la lógica del mecanismo de traducción de direcciones, y la Figura 8.23 muestra los formatos de las direcciones efectivas, de un elemento de la tabla de páginas, y de las direcciones reales. Finalmente, la Tabla 8.6 define los parámetros de un elemento de la tabla de páginas.

El esquema de gestión de memoria de 64 bits está designado para ser compatible ascendentemente con la implementación de 32 bits. En esencia, todas las direcciones efectivas, los registros

Tabla 8.6. Parámetros de gestión de memoria en el PowerPC.

Elementos de la Tabla de Segmentos

Identificador (ID) de segmento efectivo
Indica uno de los 64G segmentos efectivos; usado para determinar el elemento en la tabla de segmentos.

Bit de elemento válido (V)
Indica si hay un dato válido en este elemento.

Bit de tipo de segmento (T)
Indica si es un segmento de memoria o de E/S.

Clave de supervisor (K_s)
Usado con el número de página virtual para determinar el elemento de la tabla de páginas.

Elementos de la tabla de páginas

Bit de elemento válido (V)
Indica si hay un dato válido en este elemento.

Identificador de función de dispersión (H, *hash*)
Indica si es un elemento de dispersión primario o secundario.

Índice de página abreviada (API, *Abbreviated Page Index*)
Utilizada para establecer la coincidencia con una dirección virtual única.

Bit de referenciado (R)
El procesador pone a 1 este bit cuando se produce una operación de lectura o escritura a la página correspondiente.

Bit de cambiado (C)
El procesador pone a 1 este bit cuando se produce una operación de escritura a la página correspondiente.

Bits WIMG
W = 0: usado con la estrategia de postescritura; W = 1: usado con la estrategia de escritura directa.
I = 0: caché no inhibida; I = 1: caché inhibida.
M = 0: memoria no compartida; M = 1: memoria compartida.
G = 0: memoria no protegida; G = 1: memoria protegida.

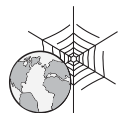
Bits de protección de página (PP)
Bits de control de acceso utilizados con los bits *K* del registro de segmento o del elemento de la tabla de segmentos para definir los derechos de acceso.

generales, y los registros de direcciones de salto se amplían a la izquierda de los 32 bits para constituir los 64 bits.

8.5. LECTURAS Y SITIOS WEB RECOMENDADOS

En [STAL98] se cubren con detalle los tópicos de este capítulo.

STAL05 Stallings, W.: *Operating Systems, Internals and Design Principles, Fifth Editon.* Upper Saddle River, NJ. Prentice Hall, 2005.



SITIOS WEB RECOMENDADOS

- **Operating System Resource Center:** una útil colección de documentos y artículos sobre una amplia gama de temas relativos a los sistemas operativos.
- **ACM Special Interest Group on Operating Systems (SIGOPS):** información acerca de las publicaciones y conferencias de este grupo de interés en sistemas operativos de la ACM (SIGOPS).
- **IEEE Technical Committee on Operating Systems and Applications:** incluye una revista *on line* y enlaces a otros sitios.
- **Revisión de sistemas operativos (Review of Operating Systems):** amplio repaso de sistemas operativos comerciales y sistemas operativos gratis, y del trabajo de investigación y entretenimiento relacionados con los sistemas operativos.

8.6. PALABRAS CLAVE, CUESTIONES Y PROBLEMAS

PALABRAS CLAVE

bloque de control de procesos	interrupción	planificación a largo plazo
buffer de traducción anticipada (*Translation Lookaside Buffer, TLB*)	lenguaje de control de trabajos (*Job Control Language, JCL*)	planificación a medio plazo
definición de particiones	memoria real	proceso
dirección física	memoria virtual	protección de memoria
dirección lógica	monitor residente	segmentación
estado del proceso	multiprogramación	sistema de colas
gestión de memoria	multitarea	sistema de tiempo compartido
hiperpaginación (*thrashing*)	núcleo (*kernel*)	sistema operativo interactivo
instrucción privilegiada	paginación	sistema operativo (*Operating System, OS*)
intercambio (*swapping*)	paginación por demanda	tabla de páginas
	planificación a corto plazo	utilidad

CUESTIONES

8.1. ¿Qué es un sistema operativo?

8.2. Enumere y defina brevemente los servicios clave que proporciona un sistema operativo.

8.3. Enumere y defina brevemente los principales tipos de planificación que realiza el sistema operativo.

8.4. ¿Cuál es la diferencia entre proceso y programa?

8.5. ¿Para qué sirve el intercambio (*swapping*)?

8.6. Si un proceso se puede asignar dinámicamente a diferentes posiciones de memoria principal, ¿qué implicaciones tiene esto en el mecanismo de direccionamiento?

8.7. ¿Es necesario que todas las páginas de un proceso estén en memoria principal mientras dicho proceso se está ejecutando?

8.8. ¿Deben estar contiguas en memoria principal las páginas de un proceso?

8.9. ¿Es necesario que las páginas de un proceso se encuentren secuencialmente ordenadas en memoria principal?

8.10. ¿Para qué sirve el buffer de traducción anticipada, TLB?

PROBLEMAS

8.1. Suponga que tenemos un computador multiprogramado en el que cada trabajo tiene características idénticas. En un periodo de computación, T, para un trabajo, la mitad del tiempo corresponde a E/S y la otra mitad a actividad de la CPU. Cada trabajo se ejecuta durante un total de N periodos. Defina las siguientes cantidades:

- Tiempo de respuesta (*Turnaround*) = tiempo para completar un trabajo.
- Rendimiento (*Throughput*) = número medio de trabajos terminados por periodo de tiempo, T.
- Utilización de CPU = porcentaje de tiempo que está activa la CPU (sin estar esperando).

Calcule estas cantidades para uno, dos y cuatro trabajos simultáneos, asumiendo que el periodo T se distribuye de cada una de las siguientes formas:

(a) E/S la primera mitad, CPU la segunda mitad.

(b) E/S el primer y el último cuarto de tiempo, CPU el segundo y el tercer cuarto.

8.2. Un programa «limitado por E/S» (*I/O bound*) es uno que, si se ejecuta en solitario, pasaría más tiempo esperando que se completen las E/S que utilizando el procesador. Un programa «limitado por el procesador» (*processor-bound*) es lo contrario. Suponga un algoritmo de planificación a corto plazo que favorece a aquellos programas que han utilizado poco el procesador en el pasado reciente. Explique por qué este algoritmo favorece a los programas limitados por E/S y sin embargo no deja sin atender a los programas limitados por el procesador.

8.3. Un programa calcula las sumas de filas

$$C_i = \sum_{j=1}^{n} a_{ij}$$

de una matriz A de 100 × 100. Asuma que el computador utiliza paginación por demanda con un tamaño de páginas de mil palabras, y que la cantidad de memoria principal reservada para datos es de cinco marcos de página. ¿Habría alguna diferencia en la fracción de faltas de página si A estuviera almacenada en memoria virtual por filas o por columnas? Explíquelo.

8.4. Considere un esquema de particiones fijas con particiones del mismo tamaño igual a 2^{16} bytes y una memoria principal de 2^{24} bytes. La tabla de procesos tiene un puntero a una partición para cada proceso residente. ¿Cuántos bits debe tener ese puntero?

8.5. Considere un esquema de particiones dinámico. Muestre que, por término medio, la memoria contiene la mitad de huecos que segmentos.

8.6. Suponga que la tabla de páginas para el proceso que se está ejecutando en un procesador es la que se muestra a continuación. Todos los números son decimales, todos se numeran desde cero y todas las direcciones de memoria son direcciones de bytes. El tamaño de la página es de 1024 bytes.

Número de página virtual	Bit de válida	Bit de referenciada	Bit de modificada	Número de marco de página
0	1	1	0	4
1	1	1	1	7
2	0	0	0	—
3	1	0	0	2
4	0	0	0	—
5	1	0	1	0

a) Describa exactamente cómo, en general, una dirección virtual generada por la CPU se traduce a una dirección física.

b) ¿Qué dirección física, si existe, correspondería a cada una de las siguientes direcciones virtuales? (no gestione ningun fallo de página, si se produce).

(i) 1052
(ii) 2221
(iii) 5499

8.7. Indique las razones por las que el tamaño de página en un sistema de memoria virtual no debe ser ni muy grande ni muy pequeño.

8.6. Un proceso hace referencia a cinco páginas, A, B, C, D, y E, en el siguiente orden:

A; B; C; D; A; B; E; A; B; C; D; E

Suponga que el algoritmo de reemplazo es del tipo primero que entre primero que sale y determine el número de transferencias de página que se producen durante esta secuencia de referencias si se comienza con una memoria principal vacía que tiene tres marcos de página. Repita el problema para cuatro marcos de página.

8.9. La siguiente secuencia de números de páginas virtuales se produce en el curso de la ejecución de un programa en un computador con memoria virtual:

3 4 2 6 4 7 1 3 2 6 3 5 1 2 3

Asuma que se ha adoptado una estrategia reemplazo de la página menos recientemente usada. Dibuje una gráfica de la tasa de aciertos de página (fracción de referencias que encuentran la página en la memoria principal) en función de la capacidad de páginas de la memoria principal, n, para $1 \leq n \leq 8$. Considere que la memoria principal está inicialmente vacía.

8.10. En el computador VAX, las tablas de páginas del usuario se sitúan en direcciones virtuales del espacio de sistema. ¿Cuál es la ventaja de tener las tablas de páginas de usuario en la memoria virtual en lugar de en la memoria principal?¿Cuál es la desventaja?

8.11. Suponga que el bucle siguiente

for (i = 1; i ⇐ n; I++)
a[i] = b[i] + c[i];

se ejecuta utilizando una memoria de mil palabras. Sea $n = 1.000$. Si se utiliza una máquina que dispone de un conjunto completo de instrucciones registro-registro y que emplea registros índice, escriba

un programa hipotético que implemente el bucle anterior. A continuación describa la secuencia de referencias a páginas durante la ejecución.

8.12. La arquitectura del IBM System/370 utiliza una estructura de memoria de dos niveles y se refiere a esos dos niveles como segmentos y páginas, aunque a la aproximación que se utiliza para la segmentación le faltan muchas de las características que se han descrito en este capítulo. Para la arquitectura del 370 básico, el tamaño de página puede ser igual a 2 KB o 4 KB, y el tamaño del segmento se fija a 64 KB o 1 MB. Para las arquitecturas del 370/XA y del 370/ESA, el tamaño de la página es de 4 KB y el del segmento 1 MB. ¿Qué ventajas de la segmentación no pueden aprovecharse con este esquema? ¿Qué beneficio aporta la segmentación del 370?

8.13. Considere un computador con segmentación y paginación. Cuando un segmento está en memoria, se desperdician algunas palabras de la última página. Además, para un segmento de tamaño s y un tamaño de página p, hay s/p elementos en la tabla de páginas. Cuanto más pequeño sea el tamaño de la página, menor será lo que se desperdicie en la última página del segmento pero mayor será la tabla de páginas. ¿Qué tamaño de página minimiza la memoria suplementaria (*overhead*) total?

8.14. Un computador tiene una caché, memoria principal, y un disco utilizado para la memoria virtual. Si una palabra está en la caché, se necesitan 20 ns para acceder a ella. Si está en memoria principal y no en la caché, se necesitan 60 ns para cargarla primero en la caché, y después empieza de nuevo la referencia a la palabra. Si la palabra no está en la memoria principal, se necesitan 12 ns para traerla del disco, más 60 ns para pasarla a la caché. La tasa de aciertos de caché es 0,9 y la tasa de aciertos de memoria principal es 0,6. ¿Cuál es el tiempo medio, en nanosegundos, que se necesita en este sistema para acceder a una palabra?

8.15. Suponga que una tarea está dividida en cuatro segmentos de igual tamaño, y que el sistema construye para cada segmento una tabla de descriptores de página con ocho elementos. Así pues, el sistema utiliza una combinación de segmentación y paginación. Asuma también que el tamaño de la página es de 2 KB.

(a) ¿Cuál es el tamaño máximo de cada segmento?
(b) ¿Cuál es el espacio de direcciones lógicas máximo para una tarea?
(c) Si una tarea accede a un elemento en la posición física 00021 ABC. ¿Cuál es el formato de la dirección lógica que la tarea genera para él? ¿Cuál es el espacio máximo de direcciones físicas del sistema?

8.16. Asuma que cierto microprocesador es capaz de acceder a 2^{32} bytes de memoria física principal. El microprocesador implementa un espacio de direcciones lógicas segmentado de un tamaño máximo de 2^{31} bytes. Cada instrucción contiene las dos partes de la dirección completa. Se utilizan unidades de gestión de memoria (MMU, *Memory Management Units*) externas, cuyo esquema de gestión de memoria asigna a los segmentos, bloques contiguos de memoria física de un tamaño fijo de 2^{22} bytes. La dirección física de comienzo de un segmento siempre es divisible por 1024. Muestre la interconexión detallada del mecanismo de correspondencia externo que convierte las direcciones lógicas en direcciones físicas utilizando el número apropiado de MMU, y muestre la estructura interna detallada de una MMU (asuma que cada MMU contiene una caché de correspondencia directa de 128 elementos para los descriptores de segmento) y la forma de seleccionar cada MMU.

8.17. Considere un espacio de direcciones lógicas paginado (compuesto por 32 páginas de 2 KB cada una) asignado a un espacio de memoria física de 1 MB.

(a) ¿Cuál es el formato de las direcciones lógicas del procesador?
(b) ¿Cuál es la longitud y la anchura de la tabla de páginas (sin considerar los bits correspondientes a los «derechos de acceso»)?
(c) ¿Qué efecto se produce en la tabla de páginas si el espacio de memoria física se reduce a la mitad?

8.18. En el sistema operativo OS/390 de IBM, uno de los módulos principales del núcleo es el gestor de recursos del sistema (*System Resource Manager*, SRM). Este módulo es el responsable de la asignación de recursos a los espacios de direcciones (procesos). El SRM proporciona al OS/390 un nivel de sofisticación único entre los sistemas operativos. Ningún otro sistema operativo para grandes computadores

(*mainframes*) y desde luego ningún otro sistema operativo, puede realizar las funciones que implementa el SRM. El concepto de recurso incluye al procesador, la memoria real y los canales de E/S. El SRM acumula datos estadísticos correspondientes a la utilización del procesador, los canales, y varias estructuras de datos clave. Se trata de alcanzar prestaciones óptimas en base a la monitorización y análisis que se realizan. Al inicializarse se establecen los objetivos a los que deben ajustarse las prestaciones y que sirven de guía para el SRM, que dinámicamente modifica la instalación y las características de ejecución de los trabajos según la utilización del sistema. Además, el SRM proporciona informes que permiten al operador experto refinar la configuración y los valores de los parámetros para mejorar el servicio que se proporciona al usuario.

Este problema se refiere a un ejemplo de la actividad del SRM. La memoria real está dividida en bloques de igual tamaño denominados marcos. Puede haber miles de ellos. Cada marco puede tener un bloque de memoria virtual denominado página. El SRM recibe el control unas veinte veces por minuto, aproximadamente, e inspecciona todas las tramas de página. Si no se ha hecho referencia a la página o no se ha cambiado, un contador se incrementa en 1. Pasado un cierto tiempo, el SRM promedia el valor de esos contadores para determinar el número medio de segundos durante el que no se toca una trama de página. ¿Para qué puede servir esto y que acción llevaría a cabo el SRM?

PARTE 3

LA UNIDAD CENTRAL DE PROCESAMIENTO

CUESTIONES A TRATAR EN LA PARTE TRES

Hemos visto hasta ahora la CPU (el procesador) esencialmente como una «caja negra» y hemos considerado su interacción con las E/S y la memoria. La Parte Tres se dedica a la estructura y funcionamiento de la CPU. La CPU consta de registros, la unidad aritmético-lógica, la unidad de ejecución de instrucciones, una unidad de control y las interconexiones entre estos componentes. Se cubren los temas de arquitectura, tales como el diseño del repertorio de instrucciones y los tipos de datos. En esta parte se tratan también aspectos relativos a organización, tales como la segmentación (*pipelining*).

ESQUEMA DE LA PARTE TRES

CAPÍTULO 9. ARITMÉTICA DEL COMPUTADOR

El Capítulo 9 examina el funcionamiento de la ALU y se centra en la representación de los números y las técnicas para realizar operaciones aritméticas. Los procesadores normalmente admiten dos tipos de aritmética: de coma fija o de enteros, y de coma flotante. Para ambos casos, el capítulo primero analiza la representación de los números y posteriormente trata las operaciones aritméticas. Se estudia con detalle el importante estándar de coma flotante IEEE 754.

CAPÍTULO 10. REPERTORIOS DE INSTRUCCIONES: CARACTERÍSTICAS Y FUNCIONES

Desde el punto de vista del programador, la mejor manera de entender el funcionamiento de un procesador es aprender el conjunto de instrucciones máquina que es capaz de ejecutar. El complejo tema del diseño de repertorios de instrucciones ocupa los Capítulos 10 y 11. El Capítulo 10 se centra en los aspectos funcionales del diseño de un repertorio de instrucciones. Este capítulo examina los tipos de funciones que se especifican mediante instrucciones del computador, concentrándose entonces en los

tipos de operandos (que especifican los datos con los que se opera) y en los tipos de operadores (que especifican las operaciones a realizar) que normalmente encontramos en los repertorios de instrucciones. Después se explica brevemente la relación entre las instrucciones del procesador y el lenguaje ensamblador.

CAPÍTULO 11. REPERTORIOS DE INSTRUCCIONES: MODOS DE DIRECCIONAMIENTO Y FORMATOS

Mientras el Capítulo 10 puede decirse que trata la semántica de los repertorios de instrucciones, el Capítulo 11 está más relacionado con la sintaxis de dichos repertorios. Concretamente, en este capítulo se ve la forma de especificar las direcciones de memoria y el formato general de las instrucciones del computador.

CAPÍTULO 12. ESTRUCTURA Y FUNCIONAMIENTO DE LA CPU

El Capitulo 12 se dedica a la estructura interna y funcionamiento del procesador. El capítulo describe el uso de registros como memoria interna de la CPU, empleando entonces todo el material visto hasta ese momento para proporcionar una revisión de la estructura y funcionamiento de la CPU. Se revisa su organización general (ALU, banco de registros, unidad de control) y se discute la organización del banco de registros. El resto del capítulo describe el funcionamiento del procesador cuando ejecuta instrucciones máquina. Se analiza el ciclo de instrucción para mostrar el funcionamiento y la interrelación de los ciclos de captación, indirección, ejecución e interrupción. Finalmente se explora con detalle la mejora de prestaciones que produce la segmentación.

CAPÍTULO 13. COMPUTADORES DE CONJUNTO REDUCIDO DE INSTRUCCIONES

En el resto de la parte tres se ve con más detalle las tendencias clave en el diseño de as CPU. El Capítulo 13 describe la aproximación asociada con el concepto de computador de conjunto reducido de instrucciones (RISC), que es una de las innovaciones más significativas en la organización y arquitectura de los computadores en los últimos años. La arquitectura RISC supuso un alejamiento drástico de la tendencia histórica en la arquitectura de los procesadores. Un análisis de esta aproximación pone de manifiesto muchos de los aspectos importantes sobre la organización y arquitectura de los computadores. Este capítulo examina las motivaciones para el uso del diseño RISC, tratando a continuación los detalles de diseño del repertorio de instrucciones de un RISC y la arquitectura de su CPU, y compara el RISC con la aproximación de repertorio de instrucciones complejo (CISC).

CAPÍTULO 14. PARALELISMO A NIVEL DE INSTRUCCIONES Y PROCESADORES SUPERESCALARES

El capítulo 14 examina un diseño innovador aun más reciente e igualmente importante: el procesador superescalar. Aunque la tecnología superescalar puede usarse en cualquier procesador, es especialmente adecuada para la arquitectura RISC. El capítulo muestra también el tema general del paralelismo a nivel de instrucciones.

CAPÍTULO 15 LA ARQUITECTURA IA-64

La arquitectura de repertorio de instrucciones IA-64 es una nueva aproximación que proporciona soporte hardware para el paralelismo a nivel de instrucciones, y que es muy distinta de la aproximación adoptada en las arquitecturas superescalares. El capítulo 15 comienza con una discusión sobre los factores que han motivado esta nueva arquitectura. Después, el capítulo muestra la organización general que da soporte a la arquitectura. El capítulo examina entonces con cierto detalle las características clave de la arquitectura IA-64 que facilitan el paralelismo de nivel de instrucciones.

CAPÍTULO 9

Aritmética del computador

PUNTOS CLAVE

- ➤ Los dos aspectos fundamentales de la aritmética del computador son la forma de representar los números (el formato binario) y los algoritmos utilizados para realizar las operaciones aritméticas básicas (suma, resta, multiplicación y división). Estas dos consideraciones se aplican tanto a la aritmética de enteros como a la de coma flotante.
- ➤ Las cantidades en coma flotante se expresan como un número (mantisa) multiplicado por una constante (base) elevada a una potencia entera (exponente). Los números en coma flotante pueden utilizarse para representar cantidades muy grandes y muy pequeñas.
- ➤ La mayoría de los procesadores implementan la norma o estándar IEEE 754 para la representación de números y aritmética en coma flotante. Esta norma define el formato de 32 bits así como el de 64 bits.

Comenzamos nuestro estudio del procesador con la unidad aritmético-lógica (ALU). Tras una breve introducción a la ALU, el capítulo se centra en el aspecto más complejo de la misma: la aritmética del computador. Las funciones lógicas que forman parte de la ALU se describen en el Capítulo 10, y la implementación de funciones lógicas y aritméticas sencillas mediante lógica digital se describen en el Apéndice B del libro.

La aritmética de un computador se realiza normalmente con dos tipos de números muy diferentes: enteros y en coma flotante. En ambos casos, la representación elegida es un aspecto de diseño crucial que trataremos en primer lugar, seguido de una discusión sobre las operaciones aritméticas.

Este capítulo incluye diversos ejemplos que se resaltan en el texto mediante recuadros sombreados.

9.1. LA UNIDAD ARITMÉTICO-LÓGICA

La ALU es la parte del computador que realiza realmente las operaciones aritméticas y lógicas con los datos. El resto de los elementos del computador (unidad de control, registros, memoria, E/S) están principalmente para suministrar datos a la ALU, a fin de que esta los procese y para recuperar los resultados. Con la ALU llegamos al estudio de lo que puede considerarse el núcleo o esencia del computador.

Una unidad aritmético-lógica, y en realidad todos los componentes electrónicos del computador, se basan en el uso de dispositivos lógicos digitales sencillos que pueden almacenar dígitos binarios y realizar operaciones lógicas booleanas elementales. El Apéndice B explora, para el lector interesado, la implementación de circuitos lógicos digitales.

La Figura 9.1 indica, en términos generales, cómo se interconecta la ALU con el resto del procesador. Los datos se presentan a la ALU en registros, y en registros se almacenan los resultados de las operaciones producidos por la ALU. Estos registros son posiciones de memoria temporal internas al

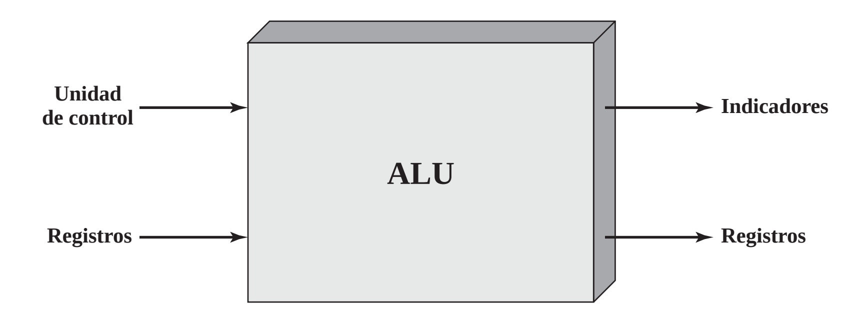


Figura 9.1. Entradas y salidas de la ALU.

procesador que están conectados a la ALU (*véase* por ejemplo la Figura 2.3). La ALU puede también activar indicadores (*flags*) como resultado de una operación.

Por ejemplo, un indicador de desbordamiento se pondrá a 1 si el resultado de una operación excede la longitud del registro en donde éste debe almacenarse. Los valores de los indicadores se almacenan también en otro registro dentro del procesador. La unidad de control proporciona las señales que gobiernan el funcionamiento de la ALU y la transferencia de datos dentro y fuera de la ALU.

9.2. REPRESENTACIÓN DE ENTEROS

En el sistema de numeración binaria[1], cualquier número puede representarse tan solo con los dígitos 1 y 0, el signo menos, y la **coma de la base** (que separa la parte entera de la decimal, el punto en los países anglosajones). Por ejemplo:

$$-1101{,}0101_2 = -13{,}3125_{10}$$

Sin embargo, para ser almacenados y procesados por un computador, no se tiene la posibilidad de disponer del signo y de la coma. Para representar los números solo pueden utilizarse dígitos 0 y 1. Si utilizáramos solo enteros no negativos, su representación sería inmediata.

Una palabra de ocho bits puede representar números desde 0 hasta 255, entre los que se encuentran:

$$\begin{aligned} 00000000 &= 0 \\ 00000001 &= 1 \\ 00101001 &= 41 \\ 10000000 &= 128 \\ 11111111 &= 255 \end{aligned}$$

[1] Para una revisión de los sistemas de numeración (decimal, binario, hexadecimal), consulte el Apéndice A.

En general, si una secuencia de n dígitos binarios $a_{n-1}a_{n-2}\dots\ a_1a_0$ es interpretada como un entero sin signo A, su valor es:

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

REPRESENTACIÓN EN SIGNO Y MAGNITUD

Existen varias convenciones alternativas para representar números enteros tanto positivos como negativos. Todas ellas implican tratar el bit más significativo (el más a la izquierda) de la palabra como un bit de signo: si dicho bit es 0 el número es positivo, y si es 1, el número es negativo.

La forma más sencilla de representación que emplea un bit de signo es la denominada representación signo-magnitud. En una palabra de n bits, los n–1 bits de la derecha representan la magnitud del entero. Por ejemplo:

$+18 = 00010010$
$-18 = 10010010$ (signo-magnitud)

El caso general puede expresarse como sigue:

Signo y magnitud $$A = \begin{cases} \sum_{i=0}^{n-2} 2^i a_i & \text{si } a_{n-1} = 0 \\ -\sum_{i=0}^{n-2} 2^i a_i & \text{si } a_{n-1} = 1 \end{cases}$$ **(9.1)**

La representación signo-magnitud posee varias limitaciones. Una de ellas es que la suma y la resta requieren tener en cuenta tanto los signos de los números como sus magnitudes relativas para llevar a cabo la operación en cuestión. Esto debiera quedar claro con la discusión de la Sección 9.3. Otra limitación es que hay dos representaciones del número 0:

$+0_{10} = 00000000$
$-0_{10} = 10000000$ (signo-magnitud)

Esto es un inconveniente porque es algo más difícil comprobar el valor 0 (una operación frecuentemente en los computadores) que si hubiera una sola representación.

Debido a estas limitaciones, raramente se usa la representación en signo-magnitud para implementar en la ALU las operaciones con enteros. En su lugar, el esquema más común es la representación en complemento a dos.

REPRESENTACIÓN EN COMPLEMENTO A DOS

Al igual que la de signo-magnitud, la representación en complemento a dos utiliza el bit más significativo como bit de signo, facilitando la comprobación de si el entero es positivo o negativo. Difiere

Tabla 9.1. Características de la representación numérica y la aritmética en complemento a dos.

Rango	-2^{n-1} hasta $2^{n-1}-1$
Número de representaciones del cero	Una
Negación	Realizar el complemento booleano de cada bit del correspondiente número positivo, y entonces sumar 1 al patrón de bits resultante visto como un entero sin signo.
Extensión de la longitud en bits	Añadir posiciones de bits a la izquierda rellenándolas con el valor del bit de signo original.
Regla de desbordamiento	Si se suman dos números con el mismo signo (ambos positivos o ambos negativos), solo se produce desbordamiento cuando el resultado tiene signo opuesto.
Regla de la sustracción	Para restar *B* de *A,* efectuar el complemento a dos de *B* y sumarlo a *A*.

de la representación signo-magnitud en la forma de interpretar los bits restantes. La Tabla 9.1 destaca las características clave de la representación y la aritmética en complemento a dos que serán elaboradas en esta sección y la siguiente.

La mayoría de los tratados sobre representación en complemento a dos se centran en las reglas para la obtención de los números negativos, sin pruebas formales de que el esquema utilizado «funcione». En su lugar, la presentación que hacemos de los números enteros en complemento a dos, en ésta y en la Sección 9.3, está basada en [DATT93], donde se sugiere que la representación en complemento a dos se entiende mejor definiéndola en términos de una suma ponderada de bits, como hicimos antes para las representaciones sin signo y en signo-magnitud. La ventaja de este tratamiento del tema está en que no queda duda alguna de si las reglas para las operaciones aritméticas con la notación en complemento a dos puedan no funcionar en algunos casos concretos.

Consideremos un entero de *n* bits, *A*, representado en complemento a dos. Si *A* es positivo, el bit de signo, a_{n-1}, es cero. Los restantes bits representan la magnitud del número de la misma forma que en la representación signo-magnitud; es decir:

$$A = \sum_{i=0}^{n-2} 2^i a_i \quad \text{para} \quad A \geq 0$$

El número cero se identifica como positivo y tiene por tanto un bit de signo 0 y una magnitud de todo ceros. Podemos ver que el rango de los enteros positivos que pueden representarse es desde 0 (todos los bits de magnitud son 0) hasta $2^{n-1} - 1$ (todos los bits de magnitud a 1). Cualquier número mayor requeriría más bits.

Ahora, para un número negativo *A* ($A < 0$), el bit de signo, a_{n-1}, es 1. Los $n - 1$ bits restantes pueden tomar cualquiera de las 2^{n-1} combinaciones. Por lo tanto, el rango de los enteros negativos que pueden representarse es desde -1 hasta -2^{n-1}. Sería deseable asignar los bits de los enteros negativos de tal manera que su manipulación aritmética pueda efectuarse de una forma directa, similar a la de los enteros sin signo. En la representación sin signo, para calcular el valor de un entero a partir de su expresión en bits, el peso del bit más significativo es $+2^{n-1}$. Como veremos en la

Sección 9.3, para una representación con bit de signo resulta que las propiedades aritméticas deseadas se consiguen si el peso del bit más significativo es (-2^{n-1}). Este es el convenio utilizado para la representación en complemento a dos, obteniéndose la siguiente expresión para los números negativos:

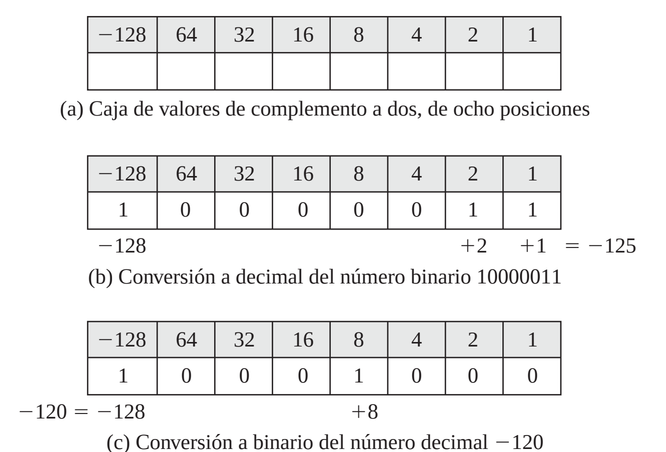
Complemento a dos $$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i \qquad \textbf{(9.2)}$$

Para los enteros positivos $a_{n-1} = 0$, de forma que el término $-2^{n-1}\,a_{n-1} = 0$. Así pues, la ecuación (9.2) define la representación en complemento a dos, tanto para los números positivos como los negativos.

La Tabla 9.2 compara, para enteros de cuatro bits, las representaciones en signo-magnitud y en complemento a dos. Veremos que la representación en complemento a dos, aunque nos pueda resultar engorrosa, facilita las operaciones aritméticas más importantes, la suma y la resta. Por esta razón, es utilizada casi universalmente como representación de los enteros en los procesadores.

Tabla 9.2. Representaciones alternativas de los enteros de 4 bits.

Representación decimal	Representación signo-magnitud	Representación complemento a dos	Representación sesgada
+8	—	—	1111
+7	0111	0111	1110
+6	0110	0110	1101
+5	0101	0101	1100
+4	0100	0100	1011
+3	0011	0011	1010
+2	0010	0010	1001
+1	0001	0001	1000
+0	0000	0000	0111
−0	1000	—	—
−1	1001	1111	0110
−2	1010	1110	0101
−3	1011	1101	0100
−4	1100	1100	0011
−5	1101	1011	0010
−6	1110	1010	0001
−7	1111	1001	0000
−8	—	1000	—



−128	64	32	16	8	4	2	1

(a) Caja de valores de complemento a dos, de ocho posiciones

−128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

−128 +2 +1 = −125

(b) Conversión a decimal del número binario 10000011

−128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

−120 = −128 +8

(c) Conversión a binario del número decimal −120

Figura 9.2. Utilización de la caja de valores para convertir entre binario en complemento a dos y decimal.

Una ilustración útil de la naturaleza de la representación en complemento a dos es una «caja» de valores, en la que el valor más a la derecha en la caja es 1 (2^0), y cada posición consecutiva hacia la izquierda tiene un valor doble a sumar (si el correspondiente bit es 1), hasta la posición más a la izquierda, cuyo valor es a restar. Como se puede ver en la Figura 9.2a, el número en complemento a dos más negativo representable es -2^{n-1}; si cualquiera de los bits distinto del de signo es 1, este añade una cantidad positiva al número. También, está claro que un número negativo debe tener un 1 en la posición más a la izquierda, y un número positivo tendrá un cero en dicha posición. Por tanto, el número positivo mayor es un 0 seguido de todo unos, que es igual a $2^{n-1} - 1$.

El resto de la Figura 9.2 ilustra el uso de la caja de valores para convertir de complemento a dos a decimal, y de decimal a complemento a dos.

CONVERSIÓN ENTRE LONGITUDES DE BITS DIFERENTES

A veces se desea tomar un entero de n bits y almacenarlo en m bits, siendo $m > n$. Esto se resuelve fácilmente en la notación signo-magnitud: simplemente trasladando el bit de signo hasta la nueva posición más a la izquierda y rellenando con ceros.

+18	=	00010010	(signo-magnitud, 8 bits)
+18	=	0000000000010010	(signo-magnitud, 16 bits)
−18	=	11101110	(signo-magnitud, 8 bits)
−18	=	1000000000010010	(signo-magnitud, 16 bits)

Este procedimiento no funciona con los enteros negativos en complemento a dos. Utilizando el mismo ejemplo:

+18	=	00010010	(complemento a dos, 8 bits)
+18	=	0000000000010010	(complemento a dos, 16 bits)
−18	=	11101110	(complemento a dos, 8 bits)
−32.658	=	1000000001101110	(complemento a dos, 16 bits)

La penúltima línea anterior puede comprobarse fácilmente mediante la caja de valores de la Figura 9.2. La última línea puede verificarse utilizando la Ecuación (9.2) o mediante una caja de valores de 16 bits.

En su lugar, la regla para los enteros en complemento a dos es trasladar el bit de signo a la nueva posición más a la izquierda y completar con copias del bit de signo. Para números positivos, rellenar con ceros, y para negativos con unos. Así pues, se tiene:

−18	=	11101110	(complemento a dos, 8 bits)
−18	=	1111111111101110	(complemento a dos, 16 bits)

Para ver que esta regla funciona, consideremos de nuevo una secuencia de *n* dígitos binarios a_{n-1} $a_{n-2} \ldots a_1 a_0$ interpretada como entero *A* en complemento a dos, tal que su valor es:

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Si *A* es positivo, la regla claramente funciona. Ahora, supongamos que *A* es negativo y que queremos construir una representación de *m* bits, con $m > n$. Entonces

$$A = -2^{m-1}a_{n-1} + \sum_{i=0}^{m-2} 2^i a_i$$

Los dos valores deben ser iguales:

$$-2^{m-1} + \sum_{i=0}^{m-2} 2^i a_i = -2^{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

$$-2^{m-1} + \sum_{i=n-1}^{m-2} 2^i a_i = -2^{n-1}$$

$$2^{n-1} + \sum_{i=n-1}^{m-2} 2^i a_i = 2^{m-1}$$

$$1 + \sum_{i=0}^{n-2} 2^i + \sum_{i=n-1}^{m-2} 2^i a_i = 1 + \sum_{i=0}^{m-2} 2^i$$

$$\sum_{i=n-1}^{m-2} 2^i a_i = \sum_{i=n-1}^{m-2} 2^i$$

$$\Rightarrow a_{m-2} = \cdots = a_{n-2} = a_{n-1} = 1$$

Al pasar de la primera a la segunda ecuación, se requiere que los $n - 1$ bits menos significativos no cambien entre las dos representaciones. Entonces, llegamos a la ecuación final, que solo es cierta si todos los bits desde la posición $n - 1$ hasta la $m - 2$ son 1. En consecuencia la regla funciona.

REPRESENTACIÓN EN COMA FIJA

Finalmente, mencionamos que la representación tratada en esta sección se denomina a veces de coma fija. Esto es porque la coma de la base (coma binaria) está fija y se supone que a la derecha del bit menos significativo. El programador puede utilizar la misma representación para fracciones binarias escalando los números de manera que la coma binaria esté implícitamente en alguna otra posición.

9.3. ARITMÉTICA CON ENTEROS

Esta sección examina funciones aritméticas comunes con números enteros representados en complemento a dos.

NEGACIÓN

En la representación signo-magnitud, la regla para obtener el opuesto de un entero es sencilla: invertir el bit de signo. En la notación de complemento a dos, la negación de un entero puede realizarse siguiendo las reglas:

1. Obtener el complemento booleano de cada bit del entero (incluyendo el bit de signo). Es decir, cambiar cada 1 por 0, y cada 0 por 1.
2. Tratando el resultado como un entero binario sin signo, sumarle 1.

Este proceso en dos etapas se denomina **transformación a complemento a dos**, u obtención del complemento a dos de un entero. Por ejemplo:

+18	=	00010010	(complemento a dos)
complemento bit a bit	=	11101101	
		+ 1	
		11101110 =	−18

Como es de esperar, el opuesto del opuesto es el propio número:

−18	=	11101110	(complemento a dos)
complemento bit a bit	=	00010001	
		+ 1	
		00010010 =	+18

Podemos demostrar la validez de la operación que acabamos de describir utilizando la definición de representación en complemento a dos dada en la Ecuación (9.2). De nuevo interpretamos una secuencia de n dígitos binarios $a_{n-1}\, a_{n-2} \ldots a_1\, a_0$ como un entero A en complemento a dos, tal que su valor es:

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Ahora se construye el complemento bit a bit, $\overline{a_{n-1}}\,\overline{a_{n-2}} \ldots \overline{a_o}$, y tratándolo como un entero sin signo, se le suma 1. Finalmente, se interpreta la secuencia resultante de n bits como un entero B en complemento a dos, de manera que su valor es

$$B = -2^{n-1}\,\overline{a_{n-1}} + 1 + \sum_{i=0}^{n-2} 2^i \overline{a_i}$$

Ahora queremos que $A = -B$, lo que significa que $A + B = 0$. Esto se comprueba fácilmente:

$$\begin{aligned} A + B &= -(a_{n-1} + \overline{a_{n-1}})\, 2^{n-1} + 1 + \left(\sum_{i=0}^{n-2} 2^i\, (a_i + \overline{a_i})\right) \\ &= -2^{n-1} + 1 + \left(\sum_{i=0}^{n-2} 2^i\right) \\ &= -2^{n-1} + 1 + (2^{n-1} - 1) \\ &= -\,2^{n-1} + 2^{n-1} = 0 \end{aligned}$$

El desarrollo anterior supone que podemos primero tratar el complemento de A bit a bit como entero sin signo al objeto de sumarle 1, y entonces tratar el resultado como un entero en complemento a dos. Hay dos casos especiales a tener en cuenta. En primer lugar, consideremos que $A = 0$. En este caso, para una representación con ocho bits:

0	=	00000000	(complemento a dos)
complemento bit a bit	=	11111111	
		+ 1	
		100000000 =	0

Hay un *acarreo* de la posición de bit más significativa, que es ignorado. El resultado es que la negación u opuesto del 0 es 0, como debe ser.

El segundo caso especial es más problemático. Si generamos el opuesto de la combinación de bits consistente en un 1 seguido de $n - 1$ ceros, se obtiene de nuevo el mismo número. Por ejemplo, para palabras de ocho bits,

−128	=	10000000	(complemento a dos)
complemento bit a bit	=	01111111	
		+ 1	
		10000000 =	−128

Esta anomalía debe evitarse. El número de combinaciones diferentes en una palabra de ocho bits es 2^n, un número par. Con ellas queremos representar enteros positivos, negativos y el 0. Cuando se representa el mismo número de enteros positivos que de negativos (en signo-magnitud) resultan dos representaciones distintas del 0. Si hay solo una representación del 0 (en complemento a dos), entonces debe haber un número desigual de números positivos que de negativos representados. En el caso del complemento a dos, hay una representación de *n* bits para el -2^{n-1}, pero no para el $+2^{n-1}$.

SUMA Y RESTA

La suma en complemento a dos se ilustra en la Figura 9.3. La suma se efectúa igual que si los números fuesen enteros sin signo. Los cuatro primeros ejemplos muestran operaciones correctas. Si el resultado de la operación es positivo, se obtiene un número positivo en forma de complemento a dos, que tiene la misma forma como entero sin signo. Si el resultado de la operación es negativo, conseguimos un número negativo en forma de complemento a dos. Obsérvese que, en algunos casos, hay un bit acarreo más allá del final de la palabra (sombreado en la figura). Este bit se ignora.

En cualquier suma, el resultado puede que sea mayor que el permitido por la longitud de palabra que está utilizando. Esta condición se denomina **desbordamiento** (*overflow*). Cuando ocurre un

1001 = −7 +0101 = 5 1110 = −2 (a) (−7) + (+5)	1100 = −4 +0100 = 4 10000 = 0 (b) (−4) + (+4)
0011 = 3 +0100 = 4 0111 = 7 (c) (+3) + (+4)	1100 = -4 +1111 = -1 11011 = -5 (d) (−4) + (−1)
0101 = 5 +0100 = 4 1001 = Desbordamiento (e) (+5) + (+4)	1001 = -7 +1010 = -6 10011 = Desbordamiento (f) (−7) + (−6)

Figura 9.3. Suma de números representados en complemento a dos.

desbordamiento, la ALU debe indicarlo para que no se intente utilizar el resultado obtenido. Para detectar el desbordamiento se debe observar la siguiente regla:

REGLA DE DESBORDAMIENTO: al sumar dos números, y ambos son o bien positivos o negativos, se produce desbordamiento si y solo si el resultado tiene signo opuesto.

Las Figuras 9.3e y f muestran ejemplos de desbordamiento. Obsérvese que el desbordamiento puede ocurrir habiéndose producido o no acarreo.

La resta se trata también fácilmente con la siguiente regla:

REGLA DE LA RESTA: para substraer un número (el substraendo) de otro (minuendo), se obtiene el complemento a dos del substraendo y se le suma al minuendo.

Así pues, la resta se consigue usando la suma, como se muestra en la Figura 9.4. Los dos últimos ejemplos demuestran que también es aplicable la regla de desbordamiento anterior.

Una ilustración gráfica como la mostrada en la Figura 9.5 [BENH92] proporciona una visión más palpable de la suma y la resta en complemento a dos. Los círculos (mitades superiores de la figura) se obtienen a partir de los correspondientes segmentos lineales de números (mitades inferiores), juntando los extremos. Observe que cuando los números se trazan en el círculo, el complemento a dos de cualquier número es el horizontalmente opuesto del mismo (indicado mediante líneas horizontales discontinuas). Comenzando en cualquier número del círculo, al sumarle un positivo k (o restarle un

```
        0010 =  2                    0101 =  5
       +1001 = –7                   +1110 = –2
        1011 = –5                   10011 =  3

(a) M = 2 = 0010              (b) M = 5 = 0101
    S = 7 = 0111                  S = 2 = 0010
   –S =     1001                 –S =     1110

        1011 = –5                    0101 = 5
       +1110 = –2                   +0010 = 2
       11001 = –7                    0111 = 7

(c) M =–5 = 1011              (d) M = 5 = 0101
    S = 2 = 0010                  S =–2 = 1110
   –S =     1110                 –S =     0010

        0111 = 7                     1010 = –6
       +0111 = 7                    +1100 = –4
        1110 = Desbordamiento       10110 = Desbordamiento

(e) M =  7 = 0111             (f) M = –6 = 1010
    S = –7 = 1001                 S =  4 = 0100
   –S =      0111                –S =      1100
```

Figura 9.4. Substracción de números en la notación de complemento a dos (M – S).

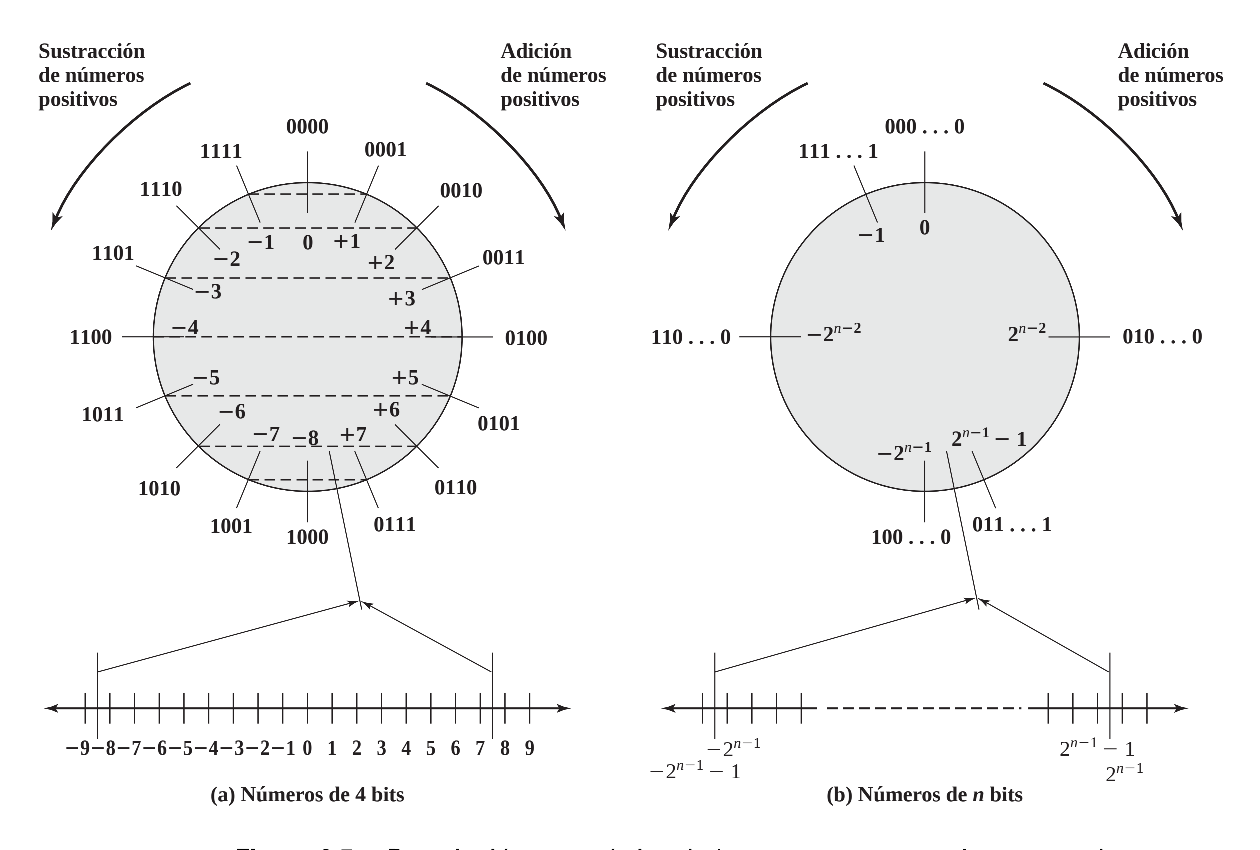


Figura 9.5. Descripción geométrica de los enteros en complemento a dos.

negativo *k*) nos desplazamos *k* posiciones en el sentido de las agujas del reloj. Restarle un positivo *k* (o sumarle un negativo *k*) equivale a desplazarse k posiciones en sentido contrario a las agujas del reloj. Si la operación realizada hace que se sobrepase el punto en que se juntaron los extremos del segmento, el resultado es incorrecto (desbordamiento).

Todos los ejemplos de las Figuras 9.3 y 9.4 pueden trazarse fácilmente en el círculo de la Figura 9.5.

La Figura 9.6 sugiere los caminos de datos y elementos hardware necesarios para realizar sumas y restas. El elemento central es un sumador binario, al que se presentan los números a sumar y produce una suma y un indicador de desbordamiento. El sumador binario trata los dos números como enteros sin signo (una implementación lógica de un sumador se da en el Apéndice B de este libro). Para sumar, los números se presentan al sumador desde dos registros, designados en este caso registros A y B. El resultado es normalmente almacenado en uno de estos registros en lugar de un tercero. La indicación de desbordamiento se almacena en un indicador o biestable de desbordamiento (OF: *Overflow Flag*) de 1 bit (0 = no desbordamiento; 1 = desbordamiento). Para la resta, el substraendo (registro B) se pasa a través de un complementador, de manera que el valor que se presenta al sumador sea el complemento a dos de B.

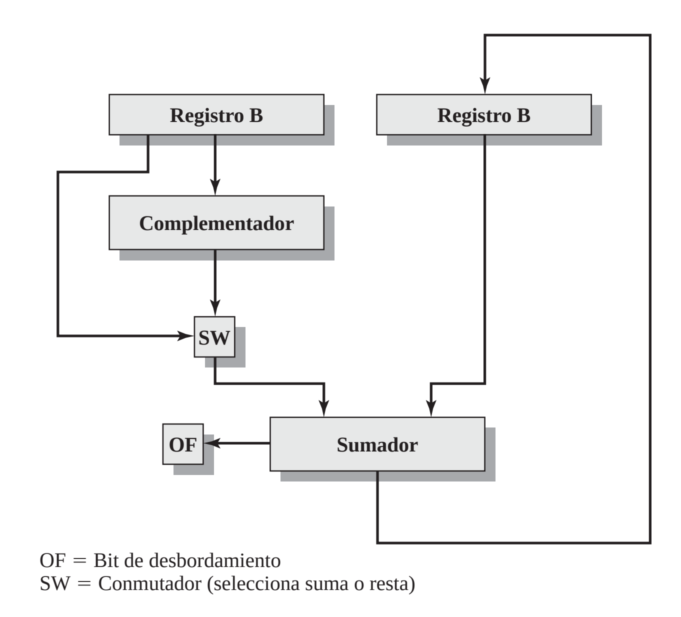


Figura 9.6. Diagrama de bloques del hardware para la suma y la resta.

MULTIPLICACIÓN

Comparada con la suma y la resta, la multiplicación es una operación compleja, ya se realice en hardware o en software. En distintos computadores se han utilizado diversos algoritmos. El propósito de esta subsección es dar al lector una idea del tipo de aproximación normalmente utilizada. Comenzaremos con el caso más sencillo de multiplicar dos enteros sin signo (no negativos), y después veremos una de las técnicas más comunes para el producto de números representados en complemento a dos.

Enteros sin signo. La Figura 9.7 ilustra la multiplicación de enteros binarios sin signo, que se realiza igual que cuando utilizamos papel y lápiz. Se pueden hacer varias observaciones:

1. La multiplicación implica la generación de productos parciales, uno para cada dígito del multiplicador. Estos productos parciales se suman después para producir el producto final.
2. Los productos parciales se definen fácilmente. Cuando el bit del multiplicador es 0, el producto parcial es 0. Cuando el multiplicador es 1, el producto parcial es el multiplicando.

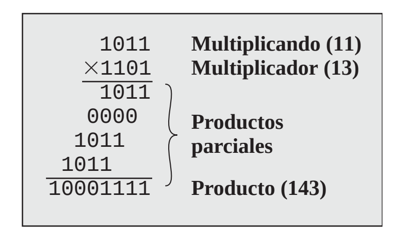


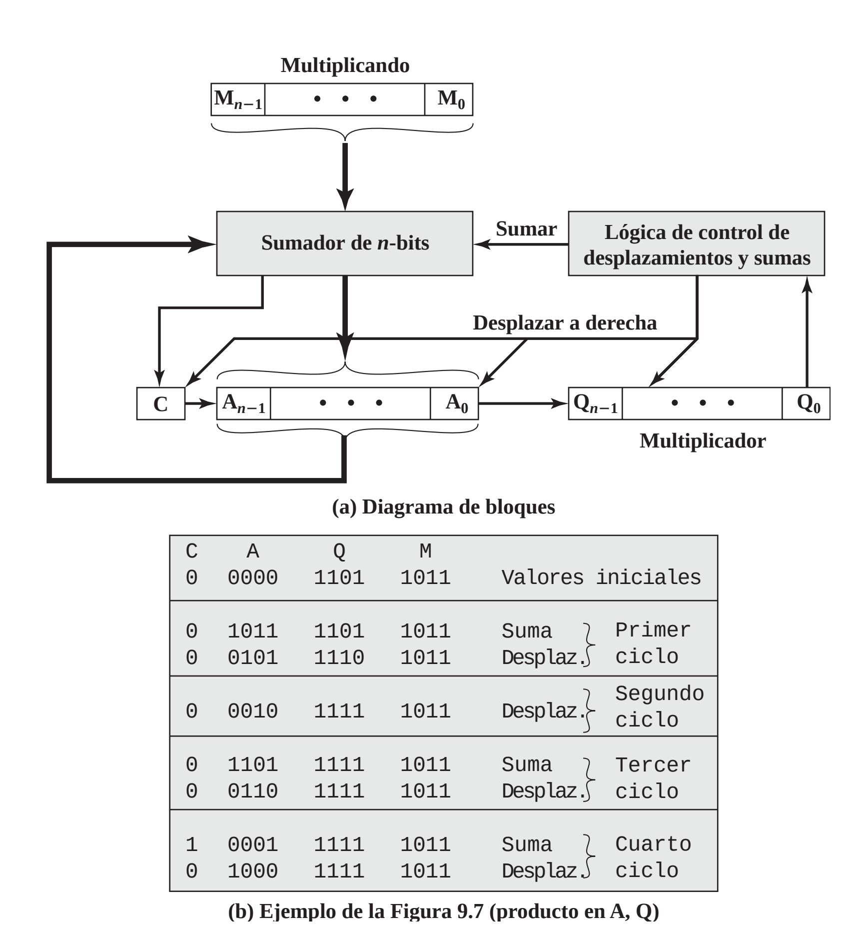
Figura 9.7. Multiplicación de enteros binarios sin signo.

3. El producto total se obtiene sumando los productos parciales. Para esta operación, cada producto parcial sucesivo se desplaza en una posición hacia la izquierda con respecto al producto parcial precedente.

4. El producto de dos enteros binarios sin signo de n bits da como resultado un producto de hasta $2n$ bits de longitud (por ejemplo, $11 \times 11 = 1001$).

En comparación con la aproximación de «papel y lápiz», hay varias modificaciones que se pueden hacer para efectuar la operación más eficientemente. En primer lugar, podemos realizar una suma progresiva de los productos parciales en lugar de esperar hasta el final. Esto evita la necesidad de almacenar todos los productos parciales, necesitándose menos registros. En segundo lugar, podemos ahorrar algún tiempo en la generación de los productos parciales. Para cada 1 del multiplicador se requiere un desplazamiento y una suma; pero por cada 0, solo se necesita el desplazamiento.

La Figura 9.8a muestra una posible implementación que hace uso de las ideas anteriores. El multiplicador y el multiplicando están ubicados en dos registros (Q y M). Un tercer registro, el registro A, es también necesario y es inicialmente puesto a 0. Hay también un registro C de un bit, inicializado a 0, que retiene los posibles bits de acarreo resultantes de las sumas.



(a) Diagrama de bloques

C	A	Q	M		
0	0000	1101	1011	Valores iniciales	
0	1011	1101	1011	Suma	Primer ciclo
0	0101	1110	1011	Desplaz.	
0	0010	1111	1011	Desplaz.	Segundo ciclo
0	1101	1111	1011	Suma	Tercer ciclo
0	0110	1111	1011	Desplaz.	
1	0001	1111	1011	Suma	Cuarto ciclo
0	1000	1111	1011	Desplaz	

(b) Ejemplo de la Figura 9.7 (producto en A, Q)

Figura 9.8. Implementación hardware de la multiplicación de binarios sin signo.

La multiplicación se efectúa de la siguiente manera. La lógica de control lee uno por uno los bits del multiplicador. Si Q_0 es 1, se suma el multiplicando al registro A y el resultado es almacenado en A, utilizando el bit C para el acarreo. Entonces se desplazan todos los bits de los registros C, A, y Q, una posición a la derecha, de manera que el bit de C pasa a A_{n-1}, A_0 pasa a Q_{n-1}, y Q_0 se pierde. Si Q_0 era 0, no se realiza la suma, solo el desplazamiento. Este proceso se repite para cada bit del multiplicador original. El producto de $2n$ bits resultante queda en los registros A y Q. La Figura 9.9 muestra un diagrama de flujo de la operación, y en la Figura 9.8b se da un ejemplo. Obsérvese que en el ciclo segundo, cuando el bit del multiplicador es 0, no hay operación de suma.

Multiplicación en complemento a dos. Hemos visto que la suma y la resta pueden realizarse con números en notación de complemento a dos, tratándolos como enteros sin signo. Consideremos:

$$\begin{array}{r} 1001 \\ +\underline{0011} \\ 1100 \end{array}$$

Si estos números se interpretan como enteros sin signo, estamos sumando 9 (1001) más 3 (0011) para obtener 12 (1100). Como enteros en complemento a dos, estamos sumando -7 (1001) a 3 (0011) para obtener -4 (1100).

Desafortunadamente, este sencillo esquema no es correcto para la multiplicación. Para verlo, consideremos de nuevo la Figura 9.7. Multiplicamos 11 (1011) por 13 (1101) para obtener 143 (10001111). Si interpretamos estos como números en complemento a dos, tendríamos -5 (1011)

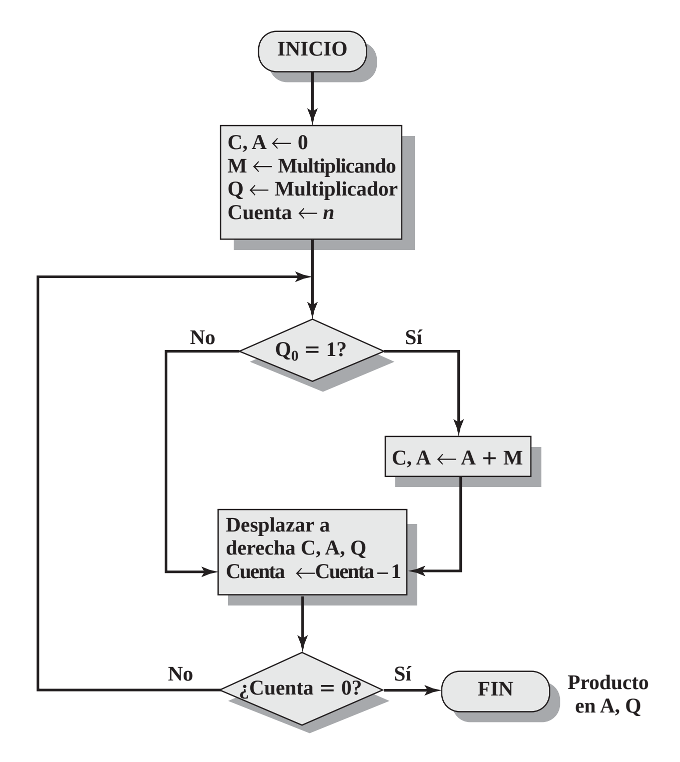


Figura 9.9. Diagrama de flujo para la multiplicación de binarios sin signo.

por -3 (1101) igual a -113 (10001111). Este ejemplo demuestra que la multiplicación directa no es adecuada si tanto el multiplicando como el multiplicador son negativos. De hecho, tampoco lo es si alguno de los dos es negativo. Para explicar este comportamiento necesitamos volver sobre la Figura 9.7 y explicar lo que se está haciendo en términos de operaciones con potencias de 2. Recuérdese que cualquier número binario sin signo puede expresarse como suma de potencias de 2. Por tanto,

$$\begin{aligned} 1101 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 2^3 + 2^2 + 2^0 \end{aligned}$$

Además, el producto de un número binario por 2^n se obtiene desplazando dicho número n bits hacia la izquierda. Teniendo esto en mente, la Figura 9.10 reestructura la Figura 9.7 para hacer la generación de productos parciales mediante multiplicación explícita. La única diferencia en la Figura 9.10 es que reconoce que los productos parciales debieran verse como números de $2n$ bits generados a partir del multiplicando de n bits.

Así pues, el multiplicando de cuatro bits 1011, como entero sin signo, es almacenado en una palabra de ocho bits como 00001011. Cada producto parcial (distinto del correspondiente a 2^0) consiste en dicho número desplazado a la izquierda, con las posiciones de la derecha rellenas con ceros (por ejemplo, un desplazamiento a la izquierda en dos posiciones produce 00101100).

Ahora podemos demostrar cómo la multiplicación directa no es correcta si el multiplicando es negativo. El problema es que cada contribución del multiplicando negativo como producto parcial tiene que ser un número negativo en un campo de $2n$ bits; los bits de signo de los productos parciales deben estar alineados. Esto se demuestra en la Figura 9.11, que muestra el producto de 1001 por 0011. Si estos se tratan como enteros sin signo se realiza el producto $9 \times 3 = 27$. Sin embargo, si 1001 se interpreta en complemento a dos como -7, cada producto parcial debe ser un número negativo en complemento a dos de $2n$ (es decir, ocho) bits, como muestra la Figura 9.11b. Obsérvese que eso podría hacerse rellenando la parte izquierda de cada producto parcial con unos.

```
    1011
  × 1101
--------
00001011   1011 × 1 × 2^0
00000000   1011 × 0 × 2^1
00101100   1011 × 1 × 2^2
01011000   1011 × 1 × 2^3
--------
10001111
```

Figura 9.10. Multiplicación de dos enteros sin signo de cuatro bits para producir un resultado de ocho bits.

```
    1001  (9)                  1001  (-7)
  × 0011  (3)                × 0011  (3)
--------                   --------
00001001  1001 × 2^0       11111001  (-7) × 2^0 = (-7)
00010010  1001 × 2^1       11110010  (-7) × 2^1 = (-14)
--------                   --------
00011011  (27)             11101011  (-21)
```

(a) Enteros sin signo (b) Enteros en complemento a dos

Figura 9.11. Comparación del producto de enteros sin signo y en complemento a dos.

Si el multiplicador es negativo, la multiplicación directa tampoco es correcta. La razón es que los bits del multiplicador ya no se corresponden con los desplazamientos o productos que deben producirse. Por ejemplo, el número decimal –3 se representa con cuatro bits en complemento a dos como 1101. Si simplemente tomamos los productos parciales basándonos en cada posición de bit, tendríamos la siguiente correspondencia:

$$1101 \longleftrightarrow -(1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0) = -(2^3 + 2^2 + 2^0)$$

De hecho, lo que se quiere es $-(2^1 + 2^0)$. Por tanto este multiplicador no puede utilizarse directamente en la forma anteriormente descrita.

Hay varias maneras de salir de este dilema. Una sería convertir tanto el multiplicando como el multiplicador en números positivos, realizar el producto y obtener después el complemento a dos del resultado si y solo si el signo de los dos números iniciales difiere. Los diseñadores han preferido utilizar técnicas que no requieren esta etapa de transformación final. Una de las técnicas más comunes es el algoritmo de Booth. Este algoritmo tiene la ventaja adicional de acelerar el proceso de multiplicación con respecto a una aproximación más directa.

El algoritmo de Booth se ilustra en la Figura 9.12 y puede describirse como sigue. Como antes, el multiplicador y el multiplicando se ubican en los registros Q y M respectivamente. Hay también un registro de un bit con una ubicación lógica a la derecha del bit menos significativo (Q_0) del registro Q, y que denominamos Q_{-1}; explicamos brevemente su uso. El producto resultante aparecerá en los

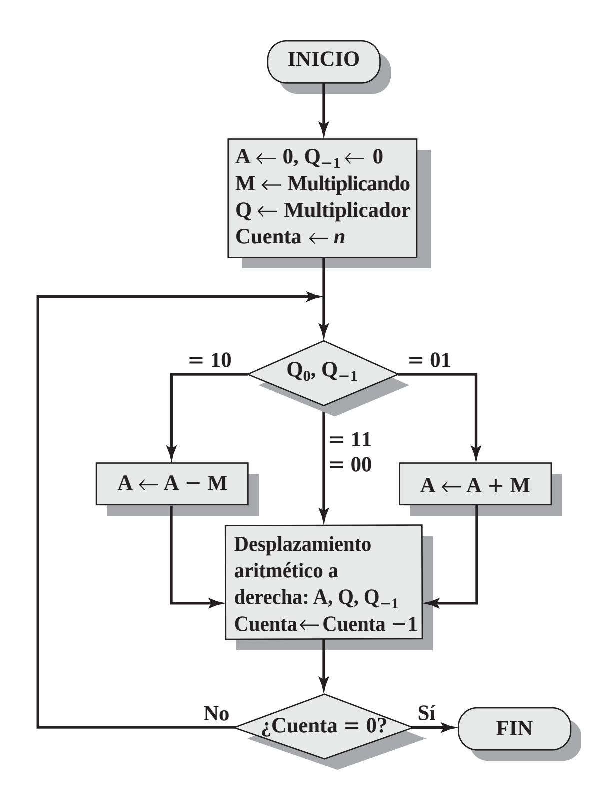


Figura 9.12. Algoritmo de Booth para la multiplicación en complemento a dos.

registros A y Q. A y Q_{-1} se fijan inicialmente a 0. Como antes, la lógica de control recorre los bits del multiplicador uno por uno. Ahora, al examinar cada bit, también se comprueba el bit a su derecha; si los dos son iguales (1-1 ó 0-0), todos los bits de los registros A, Q, y Q_{-1} se desplazan un bit a la derecha. Si dichos bits difieren, el multiplicando se suma o se resta al registro A, según que los dos bits sean 0-1 ó 1-0. A continuación de la suma o resta se realiza un desplazamiento a la derecha. En cualquier caso, el desplazamiento a la derecha es tal que el bit más a la izquierda de A, es decir A_{n-1}, no solo se desplaza a A_{n-2}, sino que también queda en A_{n-1}. Esto es necesario para preservar el signo del número contenido en la pareja de registros A y Q. Este desplazamiento se denomina **desplazamiento aritmético**, ya que preserva el bit de signo.

La Figura 9.13 muestra la secuencia de eventos para multiplicar siete por tres con el algoritmo de Booth. La misma operación se describe de manera más compacta en la Figura 9.14a.

El resto de la Figura 9.14 da otros ejemplos del algoritmo. Como puede verse actúa correctamente con cualquier combinación de números positivos y negativos. Obsérvese también la eficiencia del algoritmo. Los bloques de unos o de ceros se saltan, con un promedio de solo una suma o resta por bloque.

A	Q	Q_{-1}	M		
0000	0011	0	0111	Valores iniciales	
1001	0011	0	0111	A ← A – M	Primer ciclo
1100	1001	1	0111	Desplazam.	
1110	0100	1	0111	Desplazam.	Segundo ciclo
0101	0100	1	0111	A ← A + M	Tercer ciclo
0010	1010	0	0111	Desplazam.	
0001	0101	0	0111	Desplazam.	Cuarto ciclo

Figura 9.13. Ejemplo de aplicación del algoritmo de Booth (7 × 3).

0111	
× 0011	(0)
11111001	1–0
0000000	1–1
000111	0–1
00010101	(21)

(a) (7) × (3) = (21)

0111	
× 1101	(0)
11111001	1–0
0000111	0–1
111001	1–0
11101011	(–21)

(b) (7) × (−3) = (−21)

1001	
× 0011	(0)
00000111	1–0
0000000	1–1
111001	0–1
11101011	(–21)

(c) (−7) × (3) = (−21)

1001	
× 1101	(0)
00000111	1–0
1111001	0–1
000111	1–0
00010101	(21)

(d) (−7) × (−3) = (21)

Figura 9.14. Ejemplos de uso del algoritmo de Booth.

¿Por qué se comporta correctamente el algoritmo de Booth? Considere primero el caso en que el multiplicador sea positivo. En particular, un multiplicador positivo consistente en un bloque de unos con ceros a ambos lados (por ejemplo 00011110). Como sabemos, la multiplicación puede obtenerse sumando adecuadamente copias desplazadas del multiplicando:

$$\begin{aligned} M \times (00011110) &= M \times (2^4 + 2^3 + 2^2 + 2^1) \\ &= M \times (16 + 8 + 4 + 2) \\ &= M \times 30 \end{aligned}$$

El número de tales operaciones puede reducirse a dos si observamos que

$$2^n + 2^{n-1} + \cdots + 2^{n-K} = 2^{n+1} - 2^{n-K} \qquad \textbf{(9.3)}$$

$$\begin{aligned} M \times (00011110) &= M \times (2^5 - 2^1) \\ &= M \times (32 - 2) \\ &= M \times 30 \end{aligned}$$

Así pues, el producto puede generarse mediante una suma y una resta del multiplicando. Este esquema se extiende a cualquier número de bloques de unos del multiplicador, incluyendo el caso en que un solo 1 es tratado como bloque.

$$\begin{aligned} M \times (01111010) &= M \times (2^6 + 2^5 + 2^4 + 2^3 + 2^1) \\ &= M \times (2^7 - 2^3 + 2^2 - 2^1) \end{aligned}$$

El algoritmo de Booth obedece a este esquema, realizando una resta cuando se encuentra el primer 1 del bloque (1-0), y una suma cuando se encuentra el final (0-1) del bloque.

Para comprobar que el mismo esquema es adecuado en el caso de un multiplicador negativo, necesitamos observar lo siguiente. Sea X un número negativo en notación de complemento a dos:

Representación de $X = \{1x_{n-2}x_{n-2} \ldots x_1 x_0\}$

Entonces el valor de X puede expresarse como sigue:

$$X = -2^{n-1} + (x_{n-2} \times 2^{n-2}) + (x_{n-3}) + \cdots + (x_1 \times 2^1) + (x_0 \times 2^0) \qquad \textbf{(9.4)}$$

El lector puede verificarlo aplicando el algoritmo a los números de la Tabla 9.2.

El bit más a la izquierda de X es 1, ya que X es negativo. Suponga que el bit 0 más a la izquierda está en la posición k-ésima. Entonces X será en la forma:

$$\text{Representación de } X = \{111 \ldots 10x_{k-1}x_{k-2} \ldots x_1x_0\} \qquad \textbf{(9.5)}$$

Entonces el valor de X es:

$$X = -2^{n-1} + 2^{n-2} + \cdots + 2^{k+1} + (x_{k-1} \times 2^{k-1}) + \cdots + (x_0 \times 2^0) \quad \textbf{(9.6)}$$

Utilizando la ecuación (9.3) podemos decir que:

$$2^{n-2} + 2^{n-3} + \cdots + 2^{k+1} = 2^{n-1} - 2^{k+1}$$

Reagrupando se tiene:

$$-2^{n-1} + 2^{n-2} + 2^{n-3} + \cdots + 2^{k+1} = -2^{k+1} \quad \textbf{(9.7)}$$

Sustituyendo la ecuación (9.7) en la (9.6) tenemos:

$$X = -2^{k+1} + (x_{k-1} \times 2^{k-1}) + \cdots + (x_0 \times 2^0) \quad \textbf{(9.8)}$$

Al fin podemos volver sobre el algoritmo de Booth. Recordando la representación de X [(Ecuación (9.5)], está claro que todos los bits desde x_0 hasta el 0 más a la izquierda son manipulados de forma apropiada, ya que producen todos los términos de la ecuación (9.8) excepto el (-2^{k+1}). Cuando el algoritmo recorre hasta el 0 más a la izquierda y encuentra el 1 siguiente (2^{k+1}), ocurre una transición 1-0 y tiene lugar una resta (-2^{k+1}). Este es el término restante de la ecuación (9.8).

Como ejemplo, considere la multiplicación de algún multiplicando por (−6). En la representación de complemento a dos, utilizando una palabra de ocho bits, (−6) se expresa como 11111010. Por la ecuación (9.4) sabemos que:

$$-6 = -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1$$

como puede verificar fácilmente el lector. Por tanto,

$$M \times (11111010) = M \times (-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1)$$

Usando la ecuación (9.7),

$$M \times (11111010) = M \times (-2^3 + 2^1)$$

que, como el lector puede verificar, es también M × (−6). Finalmente, siguiendo nuestra línea de razonamiento anterior:

$$M \times (11111010) = M \times (-2^3 + 2^2 - 2^1)$$

Podemos ver que el algoritmo de Booth se ajusta a este esquema. Realiza una resta cuando se encuentra el primer 1, (1-0), una suma cuando se encuentra (0-1), y finalmente resta otra vez cuando encuentra el primer 1 del siguiente bloque de unos. Por consiguiente, el algoritmo de Booth realiza menos sumas y restas que un algoritmo directo.

DIVISIÓN

La división es algo más compleja que la multiplicación pero está basada en los mismos principios generales. Como antes, la base para el algoritmo es la aproximación de «papel y lápiz», y la operación conlleva repetidos desplazamientos y sumas o restas.

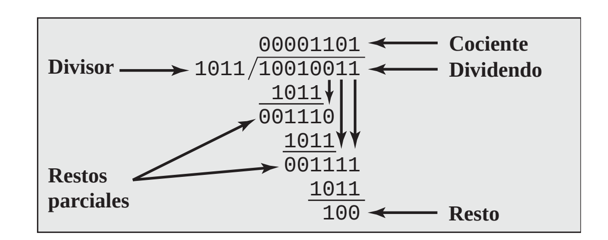


Figura 9.15. División de enteros binarios sin signo.

La Figura 9.15 muestra un ejemplo de división larga de enteros binarios sin signo. Es instructivo describir en detalle el proceso. Primero se examinan los bits del dividendo de izquierda a derecha hasta que el conjunto de bits examinados represente un número mayor o igual que el divisor; o, en otras palabras, hasta que el divisor sea capaz de dividir al número. Hasta que eso ocurre, se van colocando ceros en el cociente de izquierda a derecha. Cuando dicho evento ocurre, se coloca un 1 en el cociente, y se substrae el divisor del dividendo parcial. Al resultado se le denomina *resto parcial*. Desde este punto en adelante, la división sigue un patrón cíclico. En cada ciclo, se añaden bits adicionales del dividendo al resto parcial hasta que el resultado sea mayor o igual que el divisor. Como antes, de este número se resta el divisor para producir un nuevo resto parcial. El proceso continúa hasta que se acaban los bits del dividendo.

La Figura 9.16 muestra un algoritmo máquina que corresponde al proceso de división larga discutido. El divisor se ubica en el registro M, y el dividendo en el registro Q. En cada paso, los registros A y Q son desplazados conjuntamente un bit a la izquierda. M es restado de A para determinar si A divide el resto parcial[2]. Si esto se cumple, entonces Q_0 se hace 1. Si no, Q_0 se hace 0, y M debe sumarse de nuevo a A para restablecer el valor anterior. La cuenta entonces se decrementa, y el proceso continúa hasta *n* pasos. Al final, el cociente queda en el registro Q y el resto en el A.

Este proceso puede, con cierta dificultad, aplicarse también a números negativos. Aquí damos una posible aproximación para números en complemento a dos. En la Figura 9.17 se muestran varios ejemplos de esta aproximación. El algoritmo puede resumirse como sigue:

1. Cargar el divisor en el registro M y el dividendo en los registros A y Q. El dividendo debe estar expresado como número en complemento a dos de 2*n* bits. Por ejemplo, el número de 4 bits 0111 pasa a ser 00000111, y el 1001 pasa a 11111001.
2. Desplazar A y Q una posición de bit a la izquierda.
3. Si M y A tienen el mismo signo, ejecutar $A \leftarrow A - M$; si no, $A \leftarrow A + M$.
4. La operación anterior tiene éxito si el signo de A es el mismo antes y después de la operación.
   a) Si la operación tiene éxito o $A = 0$, entonces hacer $Q_0 \leftarrow 1$.
   b) Si la operación no tiene éxito y $A \neq 0$, entonces $Q_0 \leftarrow 0$, y restablecer el valor anterior de A.

[2] Es una resta de enteros sin signo. Si se produce un acarreo negativo (adeudo) a partir del bit más significativo, el resultado es negativo.

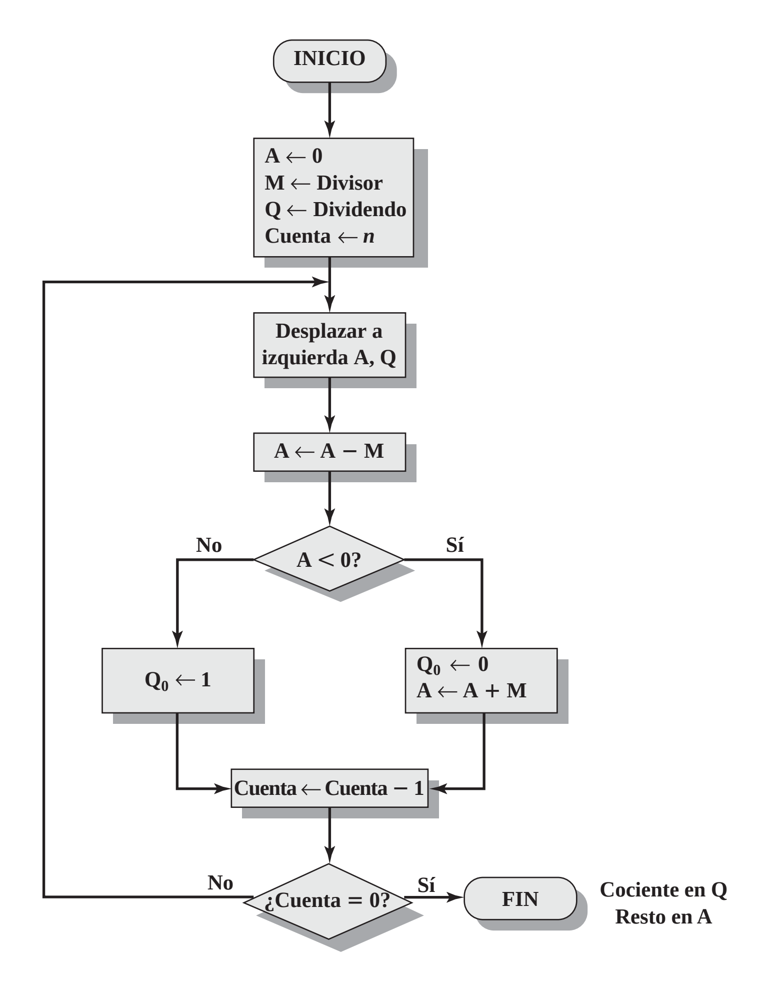


Figura 9.16. Diagrama de flujo para la división de binarios sin signo.

5. Repetir los pasos 2 a 4 tantas veces como número de bits tenga Q.
6. El resto está en A. Si los signos del divisor y el dividendo eran iguales, el cociente está en Q; si no, el cociente correcto es el complemento a dos de Q.

El lector notará en la Figura 9.17 que (–7)/(3) y (7)/(–3) producen restos diferentes. Esto es debido a que el resto se define como:

$$D = Q \times V + R$$

en donde:

D = dividendo
Q = cociente
V = divisor
R = resto

Los resultados de la Figura 9.17 son consistentes con dicha fórmula.

A	Q	M = 0011
0000	0111	Valor inicial
0000	1110	Desplazamiento
1101		Restar
0000	1110	Restablecer
0001	1100	Desplazamiento
1110		Restar
0001	1100	Restablecer
0011	1000	Desplazamiento
0000		Restar
0000	1001	Poner $Q_0 = 1$
0001	0010	Desplazamiento
1110		Restar
0001	0010	Restablecer

(a) (7)/(3)

A	Q	M = 1101
0000	0111	Valor inicial
0000	1110	Desplazamiento
1101		Sumar
0000	1110	Restablecer
0001	1100	Desplazamiento
1110		Sumar
0001	1100	Restablecer
0011	1000	Desplazamiento
0000		Sumar
0000	1001	Poner $Q_0 = 1$
0001	0010	Desplazamiento
1110		Sumar
0001	0010	Restablecer

(b) (7)/(−3)

A	Q	M = 0011
1111	1001	Valor inicial
1111	0010	Desplazamiento
0010		Sumar
1111	0010	Restablecer
1110	0100	Desplazamiento
0001		Sumar
1110	0100	Restablecer
1100	1000	Desplazamiento
1111		Sumar
1111	1001	Poner $Q_0 = 1$
1111	0010	Desplazamiento
0010		Sumar
1111	0010	Restablecer

(c) (−7)/(3)

A	Q	M = 1101
1111	1001	Valor inicial
1111	0010	Desplazamiento
0010		Restar
1111	0010	Restablecer
1110	0100	Desplazamiento
0001		Restar
1110	0100	Restablecer
1100	1000	Desplazamiento
1111		Restar
1111	1001	Poner $Q_0 = 1$
1111	0010	Desplazamiento
0010		Restar
1111	0010	Restablecer

(d) (−7)/(−3)

Figura 9.17. Ejemplos de división en complemento a dos.

9.4. REPRESENTACIÓN EN COMA FLOTANTE

FUNDAMENTOS

Con una notación de coma fija (por ejemplo, la representación en complemento a dos) es posible representar un rango de enteros positivos y negativos centrado en el cero. Asumiendo una coma binaria fija, dicho formato permite también representar números con parte fraccionaria.

La aproximación anterior tiene limitaciones. Los números muy grandes no pueden representarse, ni tampoco las fracciones muy pequeñas. Además, en la división de dos números grandes puede perderse la parte fraccionaria del cociente.

Para números decimales, esta limitación se supera utilizando la notación científica. Así, 976.000.000.000.000 puede representarse como $9{,}76 \times 10^{14}$, y 0,0000000000000976 puede expresarse como $9{,}76 \times 10^{-14}$. Lo que se ha hecho es mover dinámicamente la coma decimal a una posición

conveniente y utilizar el exponente del diez para mantener registrada la posición de la coma. Esto permite representar un rango de números muy grandes y muy pequeños con solo unos cuantos dígitos.

Esa misma técnica puede aplicarse a los números binarios. Podemos representar un número en la forma

$$\pm S \times B^{\pm E}$$

Este número puede almacenarse en una palabra binaria con tres campos:

- Signo: más o menos
- Parte significativa o mantisa S (del término inglés *significand*)
- Exponente E

La **base** B está implícita y no necesita memorizarse ya que es la misma para todos los números. Normalmente se supone que la coma de la base está a la derecha del bit más a la izquierda, o más significativo, de la mantisa. Es decir, se asume que hay un bit a la izquierda de la coma de la base.

Las reglas utilizadas en la representación de números binarios en coma flotante se explican mejor con un ejemplo. La Figura 9.18a muestra un formato típico de coma flotante de 32 bits. El bit más a la izquierda contiene el **signo** del número (0 = positivo, 1 = negativo). El valor del **exponente** se almacena en los ocho bits siguientes. La representación utilizada se conoce con el término de **representación sesgada**. Un valor fijo, llamado sesgo, se resta de este campo para conseguir el valor del exponente verdadero. Normalmente, el sesgo tiene un valor $(2^{k-1} - 1)$, donde k es el número de bits en el exponente binario. En este caso, un campo de ocho bits comprende números entre 0 y 255. Con un sesgo de 127, $(2^7 - 1)$, los valores verdaderos de exponente varían en el rango –127 a +128. En este ejemplo, se supone la base 2.

La Tabla 9.2 mostraba la representación sesgada para enteros de cuatro bits. Observe que cuando los bits de una representación sesgada se tratan como enteros sin signo, las magnitudes relativas de los números no cambian. Por ejemplo, tanto en la representación sin signo como en la sesgada, el número más grande es el 1111 y el más pequeño el 0000. Esto no se cumple para las representaciones en signo-magnitud y en complemento a dos. Una ventaja de la representación sesgada es que los números en coma flotante no negativos pueden ser tratados de la misma forma que los enteros al efectuar comparaciones.

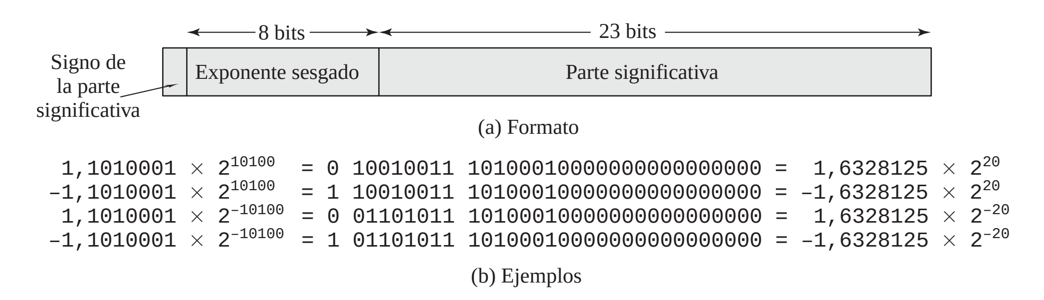


Figura 9.18. Formato típico de 32 bits en coma flotante.

La parte final de la palabra (23 bits en el ejemplo en este caso) es la **parte significativa**[3].

Cualquier número en coma flotante puede expresarse de distintas formas.

Las siguientes representaciones, con la parte significativa expresada en forma binaria, son equivalentes:

$$0{,}110 \times 2^5$$
$$110 \times 2^2$$
$$0{,}0110 \times 2^6$$

Para simplificar los cálculos con números en coma flotante se requiere usualmente que estén normalizados. En un número normalizado, el dígito más significativo de la parte significativa es distinto de cero. Por lo tanto, para la representación en base 2, el bit más significativo de la parte significativa de un número normalizado debe ser uno. Como habíamos mencionado, es una convención usual que se tenga un bit a la izquierda de la coma de la base.

Por lo tanto, un número normalizado en base dos tiene la forma:

$$\pm 0{,}1bbb \ldots b \times 2^{\pm E}$$

en donde cada b es un dígito binario (1 ó 0). Ya que el bit más significativo es siempre un 1, es innecesario almacenar este bit, está implícito. Así, el campo de 23 bits se emplea para albergar una parte significativa de 24 bits con un valor en el intervalo abierto [1, 2). Un número que no esté normalizado puede normalizarse desplazando la coma de la base hasta que quede a la derecha del bit más a la izquierda, y ajustando convenientemente el exponenente.

La Figura 9.18b da algunos ejemplos de números almacenados en este formato. Observe las siguientes características:

- El signo se almacena en el primer bit de la palabra.
- El primer bit de la parte significativa original siempre es 1 y no necesita almacenarse en el campo asignado a la parte significativa.
- Se suma 127 al exponente original para almacenarlo en el campo de exponente.
- La base es 2.

La Figura 9.19 compara los rangos de números que pueden representarse en una palabra de 32 bits. Usando la notación entera en complemento a dos, pueden representarse todos los enteros desde -2^{31} hasta $2^{31} - 1$, con un total de 2^{32} números diferentes. Con el ejemplo de formato en coma flotante de la Figura 9.18 son posibles los siguientes rangos de números:

[3] El autor considera que el término *mantisa*, que suele usarse en lugar de parte *significativa*, está obsoleto. La mantisa representa también la parte fraccionaria de un logaritmo. Por eso, salvo ocasiones en que pueda haber confusión en el texto, evitaremos usar el término.

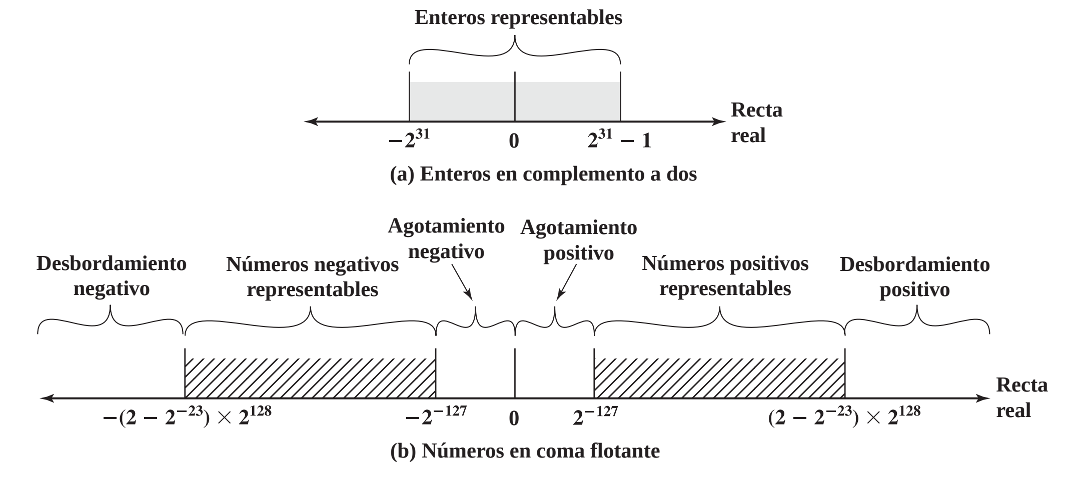


Figura 9.19. Números representables en formatos típicos de 32 bits.

- Números negativos entre $-(2 - 2^{-23}) \times 2^{128}$ y -2^{-127}
- Números positivos entre 2^{-127} y $(2 - 2^{-23}) \times 2^{128}$

En la recta de los números reales hay cinco regiones excluidas de dichos rangos:

- Los números negativos menores que $-(2 - 2^{-23}) \times 2^{128}$, región denominada **desbordamiento negativo**.
- Los números negativos mayores que 2^{-127}, denominada **agotamiento negativo**.
- El cero.
- Los números positivos menores que 2^{-127}, región denominada **agotamiento positivo**.
- Los números positivos mayores que $(2 - 2^{-23}) \times 2^{128}$, denominada **desbordamiento positivo**.

La representación indicada no contempla un valor para el 0. Sin embargo, como veremos, en la práctica se incluye una combinación de bits especial para designar el cero. Un desbordamiento ocurre cuando una operación aritmética da lugar a un número cuyo exponente es mayor que 128 (por ejemplo, $2^{120} \times 2^{100} = 2^{220}$ produciría desbordamiento). Un agotamiento ocurre cuando una magnitud fraccionaria es demasiado pequeña (por ejemplo, $2^{-120} \times 2^{-100} = 2^{-220}$). Un agotamiento es un problema menos serio porque el resultado puede generalmente aproximarse satisfactoriamente por 0.

Es importante observar que con la notación en coma flotante no estamos representando más valores individuales. El máximo número de valores diferentes que pueden representarse con 32 bits es 2^{32}. Lo que hemos hecho es repartir dichos números a lo largo de dos intervalos, uno positivo y otro negativo.

Obsérvese también que, al contrario de lo que ocurre con los números en coma fija, los números representados en la notación de coma flotante no están espaciados por igual a lo largo de la recta real.

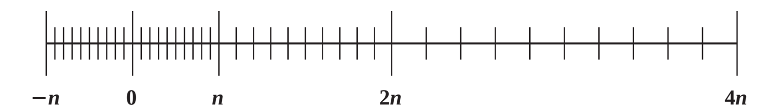


Figura 9.20. Densidad de los números en coma flotante.

Los posibles valores están más próximos cerca del origen y más separados a medida que nos alejamos de él, según se muestra en la Figura 9.20. Este es uno de los inconvenientes del cálculo en coma flotante: muchos cálculos producen resultados que no son exactos y tienen que redondearse al valor más próximo representable con la notación.

En el formato indicado en la Figura 9.18 existe un compromiso entre rango y precisión. El ejemplo muestra ocho bits dedicados al exponente y 23 bits a la parte significatva. Si incrementamos el número de bits del exponente expandimos el rango de números representables. Pero ya que solo puede expresarse un número dado de valores diferentes, habríamos reducido la densidad de dichos números y en consecuencia la precisión. La única forma de incrementar tanto el rango como la precisión es utilizar más bits. Así, la mayoría de los computadores ofrecen la posibilidad de utilizar, al menos, números de precisión simple y números en doble precisión. Por ejemplo, un formato de precisión simple podría ser de 32 bits, y uno de precisión doble de 64 bits.

Existe pues un compromiso entre el número de bits del exponente y el de la parte significativa. En realidad se trata de algo más complicado, ya que la base del exponente no tiene por qué ser dos. Por ejemplo, la arquitectura IBM S/390 utilizaba la base 16 [ANDE67b]. El formato consiste en un exponente de 7 bits y una parte significativa de 24 bits.

Con el formato en base 16 de IBM,

$$0{,}11010001 \times 2^{10100} = 0{,}11010001 \times 16^{101}$$

y el exponente que se almacena representa el 5 en lugar del 20.

La ventaja de utilizar una base mayor es que, para el mismo número de bits de exponente, se puede conseguir un rango de números mayor. Pero recuerde que con ello no incrementamos el número de valores diferentes que pueden ser representados. Así, para un formato dado, una base del exponente mayor proporciona un rango mayor a costa de menor precisión.

ESTÁNDAR DEL IEEE PARA LA REPRESENTACIÓN BINARIA EN COMA FLOTANTE

La representación en coma flotante más importante es la definida en la norma o estándar 754 del IEEE, y adoptada en 1985. Este estándar se desarrolló para facilitar la portatilidad o transferencia de los programas de un procesador a otro y para alentar el desarrollo de programas numéricos sofisticados. Este estándar ha sido ampliamente adoptado y se utiliza prácticamente en todos los procesadores y los coprocesadores aritméticos actuales.

El estándar del IEEE define tanto el formato simple de 32 bits como el doble de 64 bits (Figura 9.21), con exponentes de ocho y once bits respectivamente. La base implícita es dos. Además defi-

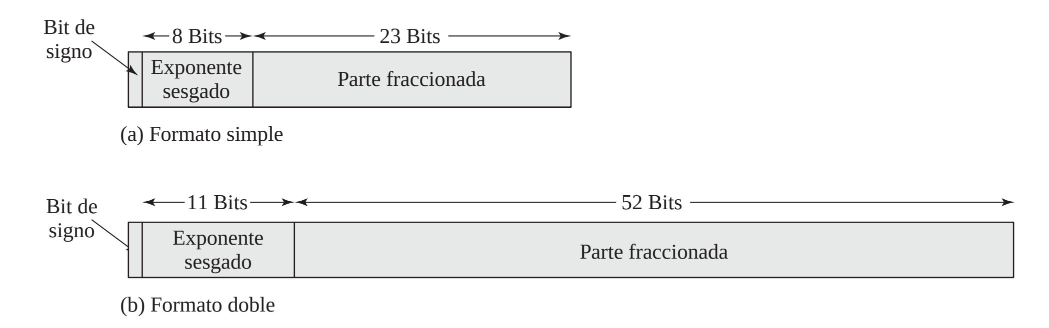


Figura 9.21. Formatos IEEE 754.

ne dos formatos ampliados, simple y doble, cuya forma exacta depende de la implementación (puede variar de unos procesadores a otros). Estos formatos ampliados incluyen bits adicionales en el exponente (rango ampliado o extendido) y en la parte significativa (precisión ampliada). Los formatos ampliados se utilizan para cálculos intermedios. Con su mayor precisión, dichos formatos reducen la posibilidad de que el resultado final se vea deteriorado por un error excesivo de redondeo; con su mayor intervalo de variación, también reducen la posibilidad de un desbordamiento intermedio que aborte un cálculo cuyo resultado habría sido representable en un formato básico. Una motivación adicional para el formato ampliado simple es que proporciona algunas de las ventajas del formato doble sin incurrir en la penalización de tiempo normalmente asociada con una precisión más elevada. La Tabla 9.3 resume las características de los cuatro formatos indicados.

En los formatos del IEEE no todos los patrones de bits se interpretan de la manera habitual. Algunas combinaciones se emplean para representar valores especiales. La Tabla 9.4 indica los valores asignados a ciertos patrones de bits. Los valores extremos de exponente consistentes en todo ceros

Tabla 9.3. Parámetros del formato IEEE 754.

Parámetro	**Formato**			
	Simple	**Simple ampliado**	**Doble**	**Doble ampliado**
Longitud de palabra (bits)	32	≥ 43	64	≥ 79
Longitud de exponente (bits)	8	≥ 11	11	≥ 15
Sesgo del exponente	127	sin especificar	1023	sin especificar
Exponente máximo	127	≥ 1023	1023	≥ 16.383
Exponente mínimo	-126	≤ -1022	-1022	≤ -16.382
Rango de números (base 10)	10^{-38}, 10^{+38}	sin especificar	10^{-308}, 10^{+308}	sin especificar
Longitud de mantisa (bits)*	23	≥ 31	52	≥ 63
Número de exponentes	254	sin especificar	2046	sin especificar
Número de fracciones	2^{23}	sin especificar	2^{52}	sin especificar
Número de valores	$1{,}98 \times 2^{31}$	sin especificar	$1{,}99 \times 2^{53}$	sin especificar

* Excluyendo el bit implícito.

Tabla 9.4. Interpretación de los números en la notación de coma flotante IEEE 754.

	Precisión simple (32 bits)				Doble precisión (64bits)			
	Signo	Exponente sesgado	Parte fraccionaria	Valor	Signo	Exponente sesgado	Parte fraccionaria	Valor
Cero positivo	0	0	0	0	0	0	0	0
Cero negativo	1	0	0	-0	1	0	0	0
Más infinito	0	255 (todo unos)	0	∞	0	2047 (todo unos)	0	∞
Menos infinito	1	255)todo unos)	0	$-\infty$	1	2047 (todo unos)	0	$-\infty$
NaN silencioso	0 ó 1	255 (todo unos)	$\neq 0$	NaN	0 ó 1	2047 (todo unos)	$\neq 0$	NaN
NaN indicador	0 ó 1	255 (todo unos)	$\neq 0$	NaN	0 ó 1	1047 (todo unos)	$\neq 0$	NaN
Positivo normalizado ≠ cero	0	$0 < e < 255$	f	$2^{e-127}(1,f)$	0	$0 < e < 2047$	f	$2^{e-1023}(1,f)$
Negativo normalizado ≠ cero	1	$0 < e < 255$	f	$-2^{e-127}(1,f)$	1	$0 < e < 2047$	f	$-2^{e-1023}(1,f)$
Positivo denormalizado	0	0	$f \neq 0$	$2^{e-126}(0,f)$	0	0	$f \neq 0$	$2^{e-1022}(0,f)$
Negativo denormalizado	1	0	$f \neq 0$	$-2^{e-126}(0,f)$	1	0	$f \neq 0$	$-2^{e-1022}(0,f)$

(0) y todo unos (255 en el formato simple, 2047 en el doble) definen valores especiales. Se representan las siguientes clases de números:

- Para valores de exponente desde 1 hasta 254 en el formato simple y desde 1 hasta 2046 en el formato doble, se representan números en coma flotante normalizados distintos de cero. El exponente está sesgado, siendo el rango de exponentes desde –126 hasta +127 en el formato simple y de (−1022) a +1023 en el doble. Un número normalizado debe contener un bit 1 a la izquierda de la coma binaria; este bit está implícito, dando una parte significativa efectiva de 24 bits o de 53 bits (denominada *fracción* o *parte fraccionaria* en el estándar).
- Un exponente cero junto con una fracción cero representa el cero positivo o el negativo, dependiendo del bit de signo. Como se mencionó, es útil tener una representación del valor 0 exacto.
- Un exponente todo unos junto con una parte fraccionaria cero representa, dependiendo del bit de signo, el infinito positivo o el negativo. Es también útil tener una representación de infinito. Esto deja al usuario decidir si trata el desbordamiento como error o si prosigue con él en el programa que se esté ejecutando.
- Un exponente cero junto con una parte fraccionaria distinta de cero representa un número denormalizado. En este caso, el bit a la izquierda de la coma binaria es cero y el exponente original es −126 ó −1022. El número es positivo o negativo dependiendo del bit de signo.
- A un exponente de todo unos junto con una fracción distinta de cero se le da el nombre de NaN, que viene de "*not a number*" (*no representa un número*), y se emplea para señalar varias condiciones de excepción.

El significado de los números denormalizados y de los NaN se discute en la Sección 9.5.

9.5. ARITMÉTICA EN COMA FLOTANTE

La Tabla 9.5 resume las operaciones básicas de la aritmética en coma flotante. En sumas y restas es necesario asegurar que ambos operandos tengan el mismo exponente. Esto puede requerir desplazar la coma de la base en uno de los operandos para conseguir alinearlos. La multiplicación y la división son más directas.

En una operación en coma flotante se puede producir alguna de estas condiciones:

- **Desbordamiento del exponente:** un exponente positivo que excede el valor de exponente máximo posible. En algunos sistemas, este desbordamiento puede designarse como $+\infty$ o $-\infty$.
- **Agotamiento del exponente:** un exponente negativo menor que el mínimo valor posible (ejemplo: -200 es menor que -127). Esto significa que el número es demasiado pequeño para ser representado, y que puede considerarse como 0.
- **Agotamiento de la parte significativa:** en el proceso de alineación o ajuste pueden perderse dígitos por la derecha de la parte significativa. Como comentaremos, se requiere efectuar algún tipo de redondeo.
- **Desbordamiento de la parte significativa:** la suma de dos mantisas del mismo signo puede producir un acarreo procedente del bit más significativo. Esto, como se explicará, puede arreglarse con un reajuste.

SUMA Y RESTA

En la aritmética de coma flotante, la suma y la resta son más complejas que la multiplicación y la división. Esto es debido a la necesidad de ajustar las partes significativas. Hay cuatro etapas básicas del algoritmo para sumar o restar:

Tabla 9.5. Números y operaciones aritméticas en coma flotante.

Números en punto flotante	Operaciones aritméticas
$X = X_s \times B^{X_E}$ $Y = Y_s \times B^{Y_E}$	$\left.\begin{array}{l} X + Y = (X_s \times B^{X_E - Y_E} + Y_s) \times B^{Y_E} \\ X - Y = (X_s - B^{X_E - Y_E} - Y_s) \times B^{Y_E} \end{array}\right\} X_E \leq Y_E$ $X \times y = (X_S \times Y_S) \times B^{X_E + Y_E}$ $\frac{X}{Y} = \left(\frac{X_S}{Y_S}\right) \times B^{X_E - Y_E}$

Ejemplos:

$X = 0{,}3 \times 10^2 = 30$
$Y = 0{,}2 \times 10^3 = 200$

$X + Y = (0{,}3 \times 10^{2-3} + 0{,}2) \times 10^3 = 0{,}23 \times 10^3 = 230$
$X - Y = (0{,}3 \times 10^{2-3} - 0{,}2) \times 10^3 = (-0{,}17) \times 10^3 = -170$
$X \times Y = (0{,}3 \times 0{,}2) \times 10^{2+3} = 0{,}06 \times 10^5 = 6.000$
$X \div Y = (0{,}3 \div 0{,}2) \times 10^{2-3} = 1{,}5 \times 10^{-1} = 0{,}15$

1. Comprobar valores cero.
2. Ajuste de partes significativas.
3. Sumar o restar las partes significativas.
4. Normalizar el resultado.

Un diagrama de flujo típico se muestra en la Figura 9.22. En ella, una descripción paso a paso resalta las funciones principales requeridas para la suma y la resta en coma flotante. Se asume un formato similar al de la Figura 9.21. Para la operación de suma o de resta, los dos operandos deben transferirse a los registros que serán utilizados por la ALU. Si el formato en coma flotante incluye un bit implícito en la parte significativa, dicho bit debe hacerse explícito para la operación.

Fase 1: comprobación de cero. Dado que la suma y la resta son idénticas excepto por el cambio de signo, el proceso comienza cambiando el signo del substraendo cuando se trata de una resta. A continuación, si alguno de los operandos es cero, se da el otro como resultado.

Fase 2: ajuste de las partes significativas. La etapa siguiente manipula los números para que los dos exponentes sean iguales.

Para ver la necesidad de la fase de ajuste, considere la siguiente suma en decimal:

$$(123 \times 10^{0}) + (456 \times 10^{-2})$$

Claramente, no podemos sumar directamente sus partes significativas. Los dígitos deben ponerse primero en posiciones equivalentes, es decir, el 4 del segundo número debe alinearse con el 3 del primero. Bajo estas condiciones, los exponentes serían iguales, que es la condición matemática para que dichos números se puedan sumar. Así:

$$(123 \times 10^{0}) + (456 \times 10^{-2}) = (123 \times 10^{0}) + (4{,}56 \times 10^{0}) = 127{,}56 \times 10^{0}$$

La alineación o ajuste se consigue o bien desplazando a la derecha el número más pequeño (incrementando su exponente) o desplazando a la izquierda el más grande. Ya que cualquiera de dichas operaciones puede hacer que se pierdan dígitos, es el menor el que se desplaza; con lo que los dígitos que se pierdan tienen una importancia relativa pequeña. El ajuste se consigue repitiendo desplazamientos en un dígito a la derecha de la parte de magnitud de la mantisa e incrementando el exponente hasta igualarlo con el otro (observe que si la base implícita es 16, un desplazamiento de un dígito equivale a desplazar 4 bits.) Si este proceso lleva a que parte significativa se hace 0, se da como resultado el otro número. Así, cuando dos números tienen exponentes muy diferentes, se pierde el menor de los números.

Fase 3: suma. A continuación se suman las dos partes significativas, teniendo en cuenta sus signos. Ya que los signos pueden diferir, el resultado puede ser 0. Existe también la posibilidad de desbordamiento de la mantisa en un dígito. Si es así, se desplaza a la derecha la parte significativa del resultado, y se incrementa el exponente. Como resultado podría producirse un desbordamiento en el exponente; esto se debe informar y la operación se detendría.

Fase 4: normalización. La etapa final normaliza el resultado. La normalización consiste en desplazar a la izquierda los dígitos de la mantisa hasta que el más significativo (1 bit, o 4 bits para exponentes de base 16) sea distinto de cero. Cada desplazamiento causa un decremento del

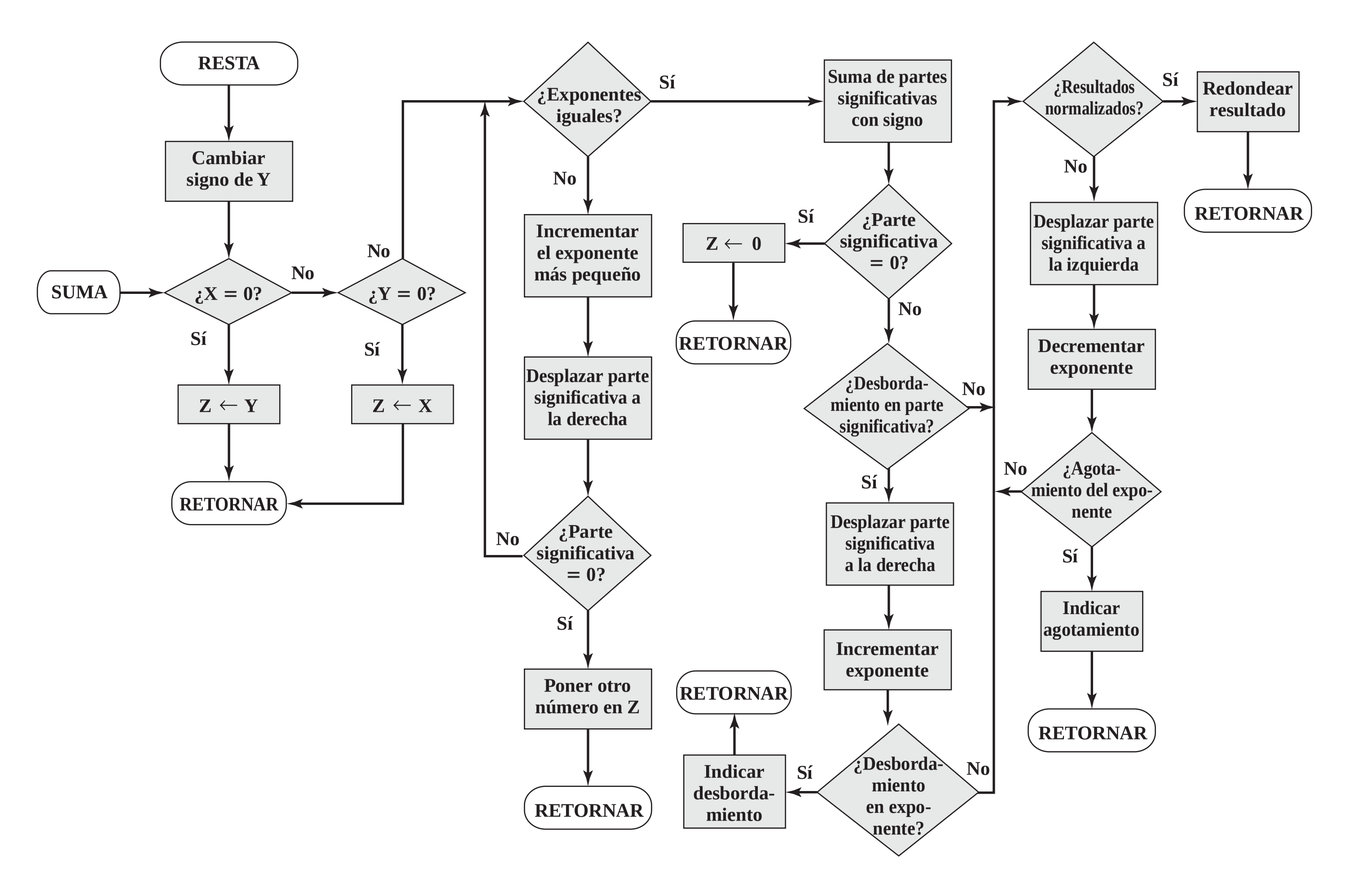


Figura 9.22. Suma y resta en coma flotante (Z ← X ± Y).

exponente, lo que podría producir un desbordamiento a cero del mismo. Finalmente, el resultado debe redondearse y proporcionarse. Posponemos el tema del redondeo para después del estudio de la multiplicación y la división.

MULTIPLICACIÓN Y DIVISIÓN

La multiplicación y la división en coma flotante son procesos mucho más sencillos que la suma y la resta, como se indica en la discusión que sigue.

Consideremos primero la multiplicación, que se ilustra en la Figura 9.23. En primer lugar, si cualquiera de los operandos es 0, se indica como resultado 0. El siguiente paso es sumar los exponentes. Si los exponentes están almacenados en forma sesgada, su suma tendría un sesgo doble. Por ello debe restarse de la suma el valor de sesgo. El resultado podría dar lugar a un desbordamiento o a un desbordamiento a cero del exponente, lo que debe indicarse, y concluir el algoritmo.

Si el exponente del producto está dentro del rango apropiado, el paso siguiente es multiplicar las partes significativas teniendo en cuenta sus signos. Dicha multiplicación se realiza como en el caso de los enteros. En este caso estamos tratando con una representación en signo-magnitud, pero los

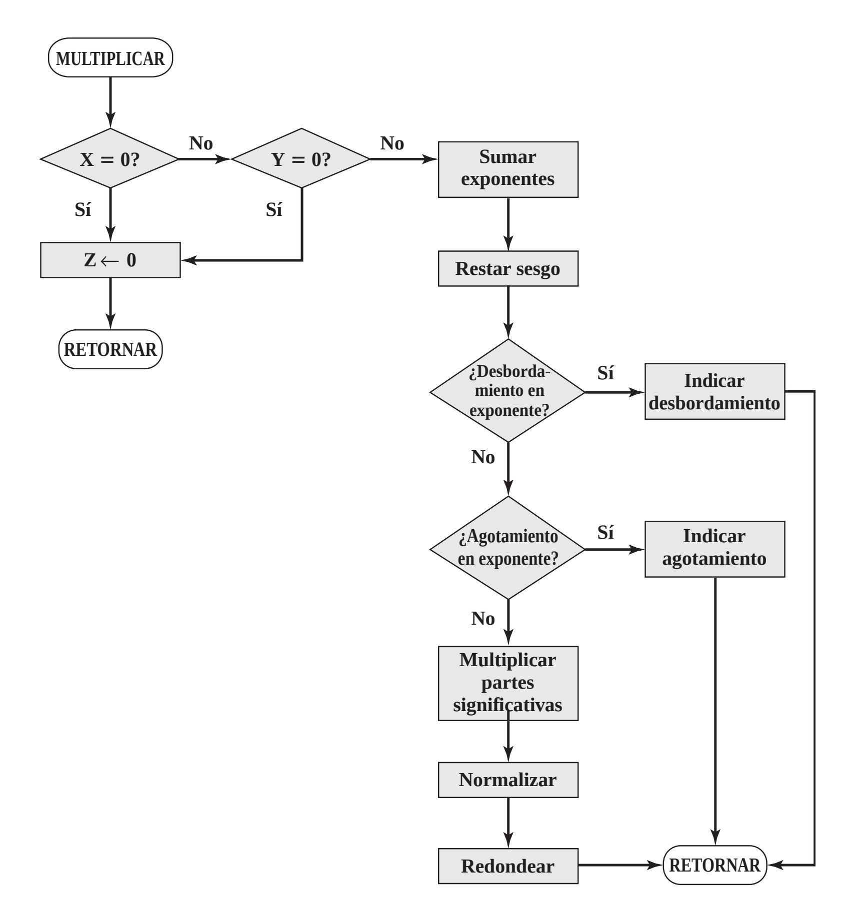


Figura 9.23. Multiplicación en coma flotante ($Z \leftarrow X \times Y$).

detalles son similares a los de representación en complemento a dos. El producto doblará la longitud del multiplicando y multiplicador. Los bits extra se perderán durante el redondeo.

Tras calcular el producto se normaliza y redondea el resultado, como se hizo para la suma y la resta. Observe que la normalización podría producir un desbordamiento a cero del exponente.

Consideremos finalmente el diagrama de flujo para la división mostrado en la Figura 9.24. De nuevo, el primer paso es comprobar ceros. Si el divisor es cero se da una indicación de error, o se pone el resultado a infinito, dependiendo de la implementación en cuestión. Un dividendo 0 da como resultado 0. A continuación el exponente del divisor se resta al del dividendo. Esto elimina el sesgo, que debe añadirse de nuevo. Se comprueban entonces posibles desbordamientos (a cero o no) del exponente.

El siguiente paso es dividir las partes significativas. A este paso sigue la normalización y redondeo usuales.

CONSIDERACIONES SOBRE PRECISIÓN

Bits de guarda. Hemos mencionado que, previo a una operación en coma flotante, se cargan el exponente y la parte significativa en registros de la ALU. En el caso de la parte significativa, el

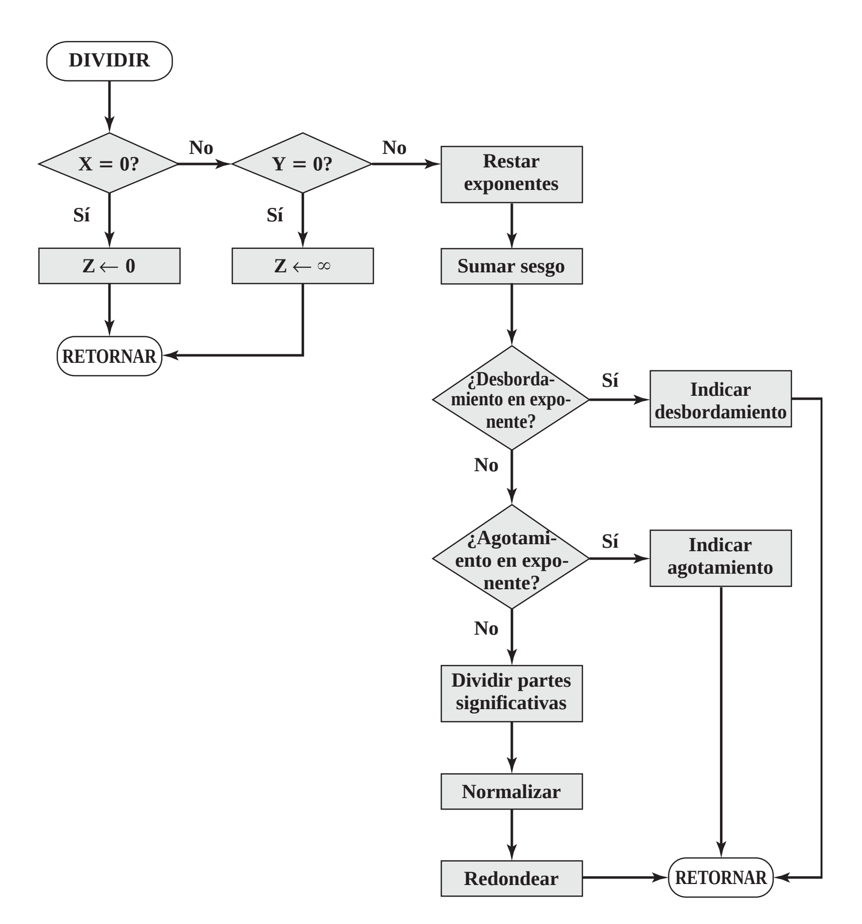


Figura 9.24. División en coma flotante (Z ← X/Y).

tamaño del registro es casi siempre mayor que la longitud de la parte significativa más el bit. El registro contiene bits adicionales, llamados bits de guarda o de respaldo, que se añaden a la derecha de la parte significativa en forma de ceros.

La razón de utilizar estos bits se ilustra en la Figura 9.25. Considere números en el formato IEEE, que tiene una parte significativa de 24 bits, incluyendo un 1 implícito a la izquierda de la coma binaria. Dos números de valor muy próximo son $X = 1,00...00 \times 2^1$ e $Y = 1,11...11 \times 2^0$. Para restar del mayor el menor de ellos, este debe desplazarse un bit a la derecha para igualar los exponentes. Esto se muestra en la Figura 9.25a. En este proceso Y pierde 1 bit significativo; el resultado que se obtiene es 2^{-22}. En la parte b de la figura se repite la misma operación pero con bits de guarda añadidos. Ahora el bit menos significativo no se pierde al alinear, y el resultado es 2^{-23}; diferente en un factor de 2 respecto del anterior resultado. Cuando la base es 16, la pérdida de precisión puede ser mayor. Como muestran las Figuras 9.25c y 9.25d, la diferencia puede ser de un factor 16.

Redondeo. Otro detalle que afecta a la precisión del resultado es la política de redondeo empleada. El resultado de cualquier operación sobre las partes significativas se almacena generalmente en un registro más grande. Cuando el resultado se pone de nuevo en el formato de coma flotante, hay que deshacerse de los bits extra.

Se han explorado diversas técnicas para realizar el redondeo. De hecho, el estándar del IEEE enumera cuatro aproximaciones alternativas:

- **Redondeo al más próximo:** el resultado se redondea al número representable más próximo.
- **Redondeo hacia +∞:** el resultado se redondea por exceso hacia más infinito.
- **Redondeo hacia −∞:** el resultado se redondea por defecto hacia menos infinito.
- **Redondeo hacia 0:** el resultado se redondea hacia cero.

Consideremos cada uno de los casos anteriores. El **redondeo al más próximo** es el modo implícito contemplado en el estándar y se define como sigue: debe tomarse el valor representable más próximo al resultado exacto.

```
 x = 1,000.....00 × 2^1
-y = 0,111.....11 × 2^1
 z = 0,000.....01 × 2^1
   = 1,000.....00 × 2^-22
```

(a) Ejemplo binario, sin bits de guarda

```
 x = ,100000 × 16^1
-y = ,0FFFFF × 16^1
 z = ,000001 × 16^1
   = ,100000 × 16^-4
```

(c) Ejemplo hexadecimal, sin bits de guarda

```
 x = 1,000.....00 0000 × 2^1
-y = 0,111.....11 1000 × 2^1
 z = 0,000.....00 1000 × 2^1
   = 1,000.....00 0000 × 2^-23
```

(b) Ejemplo binario, con bits de guarda

```
 x = ,100000 00 × 16^1
-y = ,0FFFFF F0 × 16^1
 z = ,000000 10 × 16^1
   = ,100000 00 × 16^-5
```

(d) Ejemplo hexadecimal, con bits de guarda

Figura 9.25. Utilización de bits de guarda.

Si los bits extra, además de los 23 bits que pueden almacenarse, son 10010, entonces la cantidad indicada por los bits extra supera la mitad del valor correspondiente a la última posición de bit representable. En este caso, la respuesta correcta es sumar 1 al último bit representable, redondeando por exceso al número siguiente representable. Considere ahora que los bits extra valen 01111. En este caso representan una cantidad menor que la mitad de la última posición de bit representable. La respuesta correcta es simplemente ignorar los bits extra (truncar), lo que tiene como efecto un redondeo por defecto al número representable precedente.

El estándar también contempla el caso especial de bits extra en la forma 10000..., en que el resultado está exactamente en la mitad de los dos valores representables posibles. Una posible solución sería truncar siempre, que es la operación más sencilla. Sin embargo, la dificultad de esta aproximación simple es que introduce un sesgo pequeño, pero acumulativo, en una secuencia de cálculos. Se necesita un método de redondeo que no introduzca esta tendencia de los resultados. Una posibilidad sería redondear por exceso o por defecto basándose en un número aleatorio, de manera que en promedio el resultado no sería sesgado. Un argumento en contra de esta aproximación es que no produciría resultados predecibles deterministas. La aproximación tomada en la norma del IEEE es forzar que el resultado sea par: si el resultado de un cálculo está exactamente en la mitad de dos números representables, el valor se redondea por exceso cuando el último bit representable actual es 1, y se deja como está si es 0.

Las dos siguientes opciones, **redondeo a más o a menos infinito**, son útiles en la implementación de una técnica conocida como aritmética de intervalos. La aritmética de intervalos proporciona un método eficiente para monitorizar y controlar errores en los cálculos en coma flotante, produciendo dos valores por cada resultado. Los dos valores se corresponden con los límites inferior y superior de un intervalo que contiene el resultado exacto. La longitud del intervalo, que es la diferencia entre los límites superior e inferior, indica la precisión del resultado. Si los extremos del intervalo no son representables, se redondean por defecto y por exceso, respectivamente. Aunque la anchura del intervalo puede variar de unas implementaciones a otras, se han diseñado diversos algoritmos al objeto de conseguir intervalos estrechos. Si el rango de variación entre los límites superior e inferior es suficientemente estrecho se obtiene un resultado bastante preciso. Si no, al menos lo sabemos y podemos realizar análisis adicionales.

La última de las técnicas especificadas en el estándar es el **redondeo hacia cero**. Consiste de hecho en un simple truncamiento: se ignoran los bits extra. Esta es ciertamente la técnica más sencilla. Sin embargo, el resultado es que la magnitud del valor truncado es siempre menor o igual que el valor original más exacto, introduciéndose un sesgo hacia cero en la operación. Este sesgo o tendencia es serio, ya que afecta a todas las operaciones para las que haya algún bit extra distinto de cero.

ESTÁNDAR IEEE PARA LA ARITMÉTICA BINARIA EN COMA FLOTANTE

El IEEE 754 va más allá de la simple definición de un formato, detallando cuestiones prácticas específicas y procedimientos para que la aritmética en coma flotante produzca resultados uniformes y predecibles, independientemente de la plataforma hardware. Uno de estos aspectos ha sido ya discutido, el redondeo. En esta subsección se ven otros tres aspectos: el infinito, los NaN, y los números sin normalizar o «denormalizados».

Infinito. Las operaciones aritméticas con infinito son tratadas como casos límite de la aritmética real, dándose la siguiente interpretación a los valores de infinito:

$$-\infty < \text{(todo número finito)} < +\infty$$

Exceptuando los casos especiales discutidos posteriormente, cualquier operación aritmética que involucre al infinito produce el resultado obvio conocido.

Por ejemplo:

$5 + (+\infty) = +\infty$	$5 \div (+\infty) = +0$
$5 - (+\infty) = -\infty$	$(+\infty) + (+\infty) = +\infty$
$5 + (-\infty) = -\infty$	$(-\infty) + (-\infty) = -\infty$
$5 - (-\infty) = +\infty$	$(-\infty) - (+\infty) = -\infty$
$5 \times (+\infty) = +\infty$	$(+\infty) - (-\infty) = +\infty$

NaN indicadores y silenciosos. Un NaN es una entidad simbólica codificada en formato de coma flotante, del que hay dos tipos: indicador y silencioso. Un NaN indicador señala una condición de operación no válida siempre que aparece como operando. Los NaN indicadores permiten representar valores de variables no inicializadas y tratamientos de tipo aritmético que no están contemplados en el estándar. Un NaN silencioso se propaga en la mayoría de las operaciones sin señalar excepción alguna. La Tabla 9.6 indica operaciones que producirían un NaN silencioso.

Obsérvese que ambos tipos de NaN tienen el mismo formato general (Tabla 9.4): un exponente con todo unos y una parte fraccionaria distinta de cero. El patrón de bits real de dicha fracción depende de cada implementación concreta; sus valores pueden utilizarse para distinguir entre NaN indicadores y silenciosos, y para especificar condiciones de excepción particulares.

Tabla 9.6. Operaciones que producen un NaN silencioso.

Operación	**NaN silencioso producido por**
Cualquiera	Cualquier operación con un NaN indicador
Suma o resta	Restas con infinito $(+\infty) + (-\infty)$ $(-\infty) + (+\infty)$ $(+\infty) - (+\infty)$ $(-\infty) - (-\infty)$
Multiplicación	$0 \times \infty$
División	$\frac{0}{0}$ ó $\frac{\infty}{\infty}$
Resto	x RE 0 ó ∞ RE y
Raíz cuadrada	$\sqrt{x}$ donde $x < 0$

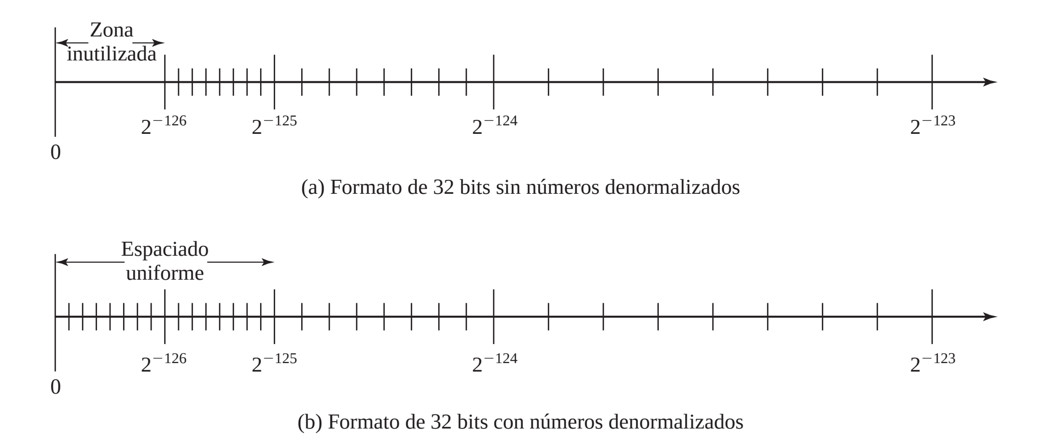


Figura 9.26. Efecto de los números denormalizados del IEEE 754.

Números denormalizados. Los números denormalizados se incluyen en el IEEE 754 para reducir las situaciones de desbordamiento hacia cero de exponentes. Cuando el exponente del resultado es demasiado pequeño (un exponente negativo con magnitud muy grande), el resultado se denormaliza desplazando a la derecha la parte fraccionaria e incrementando el exponente, a cada desplazamiento, hasta que dicho exponente esté dentro de un rango representable.

La Figura 9.26 ilustra el efecto que tiene añadir los números denormalizados. Los números que son representables pueden agruparse en intervalos de la forma $[2^n, 2^{n+1}]$. Dentro de cada intervalo, la parte de exponente del número es la misma y varía la parte fraccionaria, produciéndose un espaciado uniforme de los números representables en el intervalo. A medida que nos aproximamos a cero, cada intervalo sucesivo es la mitad de ancho que el precedente pero contiene la misma cantidad de números representables. Por tanto, la densidad de números representables aumenta a medida que nos aproximamos a cero. Sin embargo, si solo se emplean números normalizados, hay una zona inutilizada entre cero y el menor de los números normalizados. En el caso del IEEE 754 hay 2^{23} números representables en cada intervalo, y el número positivo más pequeño representable es 2^{-126}. Al tener en cuenta también los números renormalizados, se añaden 2^{23} números uniformemente distribuidos entre 0 y 2^{-126}.

El uso de números denormalizados se denomina *desbordamiento hacia cero gradual* [COON81]. Sin números denormalizados, la zona entre cero y el número distinto de cero más pequeño representable es mucho más grande que la zona entre dicho número y el que le sigue. El desbordamiento a cero gradual cubre esta zona y reduce el efecto de desbordamiento a cero del exponente hasta un nivel comparable con el redondeo de números normalizados.

9.6. LECTURAS Y SITIOS WEB RECOMENDADOS

[ERCE04] y [PARH00] son excelentes tratados sobre la aritmética de los computadores, cubriendo con detalle todos los aspectos discutidos en este capítulo. [FLYN01] proporciona una discusión útil centrada en aspectos prácticos de diseño e implementación. Para un estudio serio de la aritmética del computador, una referencia muy útil es el trabajo de dos volúmenes [SWAR90]. El Volumen I fue inicialmente publicado en 1980 y contiene artículos clave (algunos de ellos difíciles de encontrar por otras vías) sobre fundamentos de aritmética del

computador. El Volumen II contiene artículos más recientes que tratan sobre aspectos teóricos, de diseño, y de implementación.

Sobre aritmética en coma flotante, el libro [GOLD91] tiene un título apropiado: *Lo que todo informático debe conocer sobre aritmética en coma flotante.* Otro excelente tratado sobre el tema se incluye en [KNUT81], que cubre también la aritmética con enteros. También merecen la pena los siguientes tratados de mayor profundidad: [OVER01, EVEN00a, OBER97a, OBER97b, SODE96]. [KUCK77] presenta una buena discusión sobre métodos de redondeo en aritmética de coma flotante. [EVEN00b] examina los métodos de redondeo referidos al IEEE 754.

[SCHW99] describe el primer procesador IBM S/390 en integrar la base 16 y la aritmética IEEE 754 en la misma unidad de coma flotante.

ERCE04 Ercegovac, M. y Lang, T.: *Digital Arithmetic.* San Francisco. Morgan Kaufmann, 2004.

EVEN00a Even, G. y Paul, W.: «On the Desing of IEEE compliant Floating-Point Units». *IEEE Transactions on Computers,* mayo, 2000.

EVEN00b Even, G. y Seidel, P.: «A Comparison of Three Rounding Algorithms for IEEE Floating-Point Multiplication». *IEEE Transactions on Computers,* julio, 2000.

FLYN01 Flyn, M. y Oberman, S.: *Advanced Computer Arithmetic Desing.* New York. Wiley 2001.

GOLD91 Goldberg, D.: «What Every Computer Scientist Should Know About Floating-Point Arithmetic». ACM Computing Surveys, marzo, 1991.

KNUT98 Knuth, D.: *The Art of Computer Programming, Volume 2: Seminumerical Algorithms.* Reading, MA: Addison-Wesley, 1998.

KUCK77 Kuck, D.; Parker, D. y Sameh, A.: "An Analysis of Rounding Methods in Floating-Point Arthmetic¨». *IEEETransactions on Computers,* julio, 1977.

OBER97a Oberman, S. y Flynn, M.: «Desing Issues in Division and Other Floating-Point Operations». *IEEE Transactions on Computers,* julio, 1977.

OBER97b Oberman, S. y Flynn, M.: «Division Algorithms and Implementations». *IEEE Transactions on Computers,* agosto, 1997.

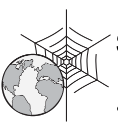
OVER01 Overton, M.: *Numerical Computing with IEEE Floating Point Arithmetic.* Philadelphia, PA. Society for Industrial and Applied Mathematics, 2001.

PARH00 Parhami, B.: *Computer Arithmetic: Algorihms and Hardware Desing.* Oxford. Oxford University Press, 2000.

SCHW99 Schwarz, E. y Krygowski, C.: «The S/390 G5 Floating-Point Unit». *IBM Journal of Research and Development,* septiembre/noviembre, 1999.

SODE96 Soderquist, P. y Leeser, M.: «Area and Performance Tradeoffs in Floating-Point Divide and Square-Root Implementations». *ACM Computing Surverys,* septiembre, 1996.

SWAR90 Swatzlander, E., editor: *Computer Arithmetic, Volumes I and II.* Los Alamitos, CA. IEEE Computer Society Press, 1990.



SITIOS WEB RECOMENDADOS

- **PCI Special Interest Group:** información acerca de las especificaciones del bus PCI y productos basados en el mismo.
- **IEEE 754:** incluye los documentos del IEEE 754, publicaciones y artículos relacionados, y múltiples enlaces útiles relacionados con la aritmética del computador.

9.7. PALABRAS CLAVE, PREGUNTAS DE REPASO Y PROBLEMAS

PALABRAS CLAVE

base	desbordamiento del exponente	producto parcial
bit de signo	desbordamiento negativo	redondeo
bits de guarda o de respaldo mantisa	desbordamiento positivo	representación en coma fija
coma de la base	desplazamiento aritmético	representación en coma flotante
desbordamiento	dividendo	representación en complemento a dos
desbordamiento a cero del exponente	divisor	representación en complemento a uno
desbordamiento a cero de la parte significativa	exponente	representación en signo-magnitud
desbordamiento a cero negativo	minuendo	representación sesgada
desbordamiento a cero positivo	multiplicador	resto
desbordamiento de la parte significativa	multiplicando	substraendo
	número normalizado	unidad aritmética y lógica (ALU)
	número renormalizado	
	parte significativa	
	producto cociente	

PREGUNTAS DE REPASO

9.1. Explique brevemente las siguientes representaciones: signo-magnitud, complemento a dos, sesgada.

9.2. Explique cómo determinar si un número es negativo en las siguientes representaciones: signo-magnitud, complemento a dos, sesgada.

9.3. ¿En qué consiste la regla de extensión del signo para los números en complemento a dos?

9.4. ¿Cómo se obtiene el opuesto de un entero en la representación de complemento a dos?

9.5. En términos generales, ¿cuándo produce el mismo entero la operación de complemento a dos sobre un entero de *n* bits?

9.6. ¿Cuál es la diferencia entre la representación en complemento a dos de un número y el complemento a dos de un número?

9.7. Si al sumar tratamos los números en complemento a dos como si fuesen enteros sin signo, el resultado, interpretado como número en complemento a dos, es correcto. ¿Por qué esto no se cumple para la multiplicación?

9.8. ¿Cuáles son los cuatro elementos esenciales de un número en la notación de coma flotante?

9.9. ¿Qué ventaja supone utilizar la representación sesgada para la parte del exponente de un número en coma flotante?

9.10. ¿Qué diferencias existen entre desbordamiento positivo, desbordamiento del exponente, y desbordamiento de la parte significativa?

9.11. ¿Cuáles son los elementos básicos de la suma y la resta en coma flotante?

9.12. Indique un motivo para el uso de los bits de guarda.

9.13. Enumere cuatro métodos alternativos de redondeo de los resultados de una operación de coma flotante.

PROBLEMAS

9.1. Represente tanto en signo-magnitud como en complemento a dos, con 16 bits, los siguientes números decimales: $+512$; -29.

9.2. Represente en decimal los siguientes valores en complemento a dos: 1101011; 0101101.

9.3. Otra representación utilizada a veces para los números enteros es el **complemento a uno.** Los enteros positivos se representan de la misma forma que en signo-magnitud. Un entero negativo se representa tomando el complemento booleano de cada bit del correspondiente número positivo.

(a) Exprese una definición de los números en complemento a uno utilizando una suma ponderada de bits, similar a las ecuaciones (9.1) y (9.2).

(b) ¿Cuál es el rango de números que puede representarse en complemento a uno?

(c) Defina un algoritmo que efectúe la suma en aritmética de complemento a uno.

9.4. Añada columnas a la Tabla 9.1 para signo-magnitud y para complemento a uno.

9.5. Considere la siguiente operación con una palabra binaria. Comenzar con el bit menos significativo. Copiar todos los bits que son 0 hasta que se encuentra el primer 1, que también se copia. A partir de este, tomar el complemento de los bits siguientes. ¿Cuál es el resultado?

9.6. En la Sección 9.3 se define la operación de complemento a dos como sigue: para calcular el complemento a dos de X, tomar el complemento booleano de cada bit de X, y después sumar 1.

(a) Compruebe que la siguiente definición es equivalente. Para un entero X de n bits, el complemento a dos de X se obtiene considerando X como entero sin signo y calculando $(2^n - X)$.

(b) Demuestre que la Figura 9.2 puede utilizarse para ilustrar gráficamente el punto anterior, mostrando cómo se usa el desplazamiento en el sentido de las agujas del reloj para realizar la substracción.

9.7. En base r, el complemento a r de un número N de n dígitos se define como: $r^n - N$ para $N \neq 0$, y 0 para $N = 0$. Calcule el complemento a diez del número decimal 13250.

9.8. Calcule $(72530 - 13250)$ empleando aritmética en complemento a diez. Suponga reglas similares a las vistas para la aritmética en complemento a dos.

9.9. Considere la suma en complemento a dos de dos números de n bits:

$$z_{n-1}z_{n\text{-}2} \ldots z_0 = x_{n-1}x \ldots x_0 + y_{n-1}y_{n-2} \ldots y_0$$

Suponga que se efectúa la suma bit a bit generandose bits de acarreo c_i por cada suma de x_i, y_i y c_{i-1}. Suponga que v es una variable binaria que se pone a uno si hay desbordamiento. Rellene los valores de la siguiente tabla.

Entrada	x_{n-1}	0	0	0	0	1	1	1	1
	y_{n-1}	0	0	1	1	0	0	1	1
	c_{n-2}	0	1	0	1	0	1	0	1
Salida	z_{n-1}								
	v								

9.10. Considere los números representados en complemento a dos con ocho bits y realice los siguientes cálculos:

(a) $6 + 13$

(b) $-6 + 13$

(c) $6 - 13$

(d) $-6 - 13$

9.11. Calcule las siguientes diferencias utilizando complemento a dos:

(a) $\begin{array}{r} 111000 \\ -\underline{110011} \end{array}$ (b) $\begin{array}{r} 11001100 \\ -\underline{101110} \end{array}$ (c) $\begin{array}{r} 111100001111 \\ -\underline{110011110011} \end{array}$ (d) $\begin{array}{r} 11000011 \\ -\underline{11101000} \end{array}$

9.12. ¿Es válida la siguiente definición alternativa de desbordamiento en aritmética de complemento a dos?

«Si la OR exclusiva de los bits de acarreo anterior y posterior a la columna más a la izquierda es 1, hay desbordamiento. En caso contrario no hay desbordamiento».

9.13. Compare las Figuras 9.9 y 9.12. ¿Por qué no se utiliza el bit C en la segunda?

9.14. Dados $x = 0101$ e $y = 1010$ en notación de complemento a dos (es decir, $x = 5$ e $y = -6$), calcule el producto $p = x \times y$ con el algoritmo de Booth.

9.15. Use el algoritmo de Booth para multiplicar 23 (multiplicando) por 29 (multiplicador), donde cada número está representado con 7 bits.

9.16. Demuestre que el producto de dos números de *n* dígitos en base B produce un resultado de no más de 2*n* dígitos.

9.17. Verifique la validez del algoritmo de división de binarios sin signo de la Figura 9.16, mostrando los pasos implicados en el cálculo de la división de la Figura 9.15. Utilice una presentación similar a la empleada en la Figura 9.17.

9.18. El algoritmo de división entera descrito en la Sección 9.3 se conoce con el nombre de método de división con restablecimiento ya que el valor del registro A debe restablecerse tras cada resta sin éxito. Una aproximación ligeramente más compleja, denominada sin restablecimiento, evita las restas y sumas innecesarias. Proponga un algoritmo para este método.

9.19. En operaciones aritméticas con enteros, el cociente *J/K* de dos enteros *J* y *K* es menor o igual que el cociente normal. ¿Verdadero o falso?

9.20. Divida -145 entre 13 en notación binaria de complemento a dos utilizando palabras de 12 bits. Emplee el algoritmo descrito en la Sección 9.3.

9.21. (a) Considere una representación de como fija que utiliza dígitos decimales, en la que la coma implícita de la base puede estar en cualquier posición (es decir, a la derecha del dígito menos significativo, a la derecha del más significativo, etc.). ¿Cuántos dígitos decimales son necesarios para representar tanto la constante de Planck ($6{,}63 \times 10^{-27}$) como el número de Avogadro ($6{,}02 \times 10^{23}$)? La coma implícita de la base debe estar en la misma posición en ambos casos.

(b) Considere ahora un formato decimal de coma flotante con el exponente almacenado en una representación sesgada, con un sesgo de 50. Se supone una representación normalizada. ¿Cuántos dígitos decimales se requieren para representar las constantes anteriores en este formato de coma flotante?

9.22. Suponga que el exponente e está restringido al rango $0 \leq e \leq X$, con un sesgo q, que la base es b, y que la parte significativa tiene una longitud de p dígitos.

(a) ¿Cuáles son los números positivos mayor y menor que pueden expresarse?

(b) ¿Cuáles son los números positivos mayor y menor que pueden expresarse como números normalizados en coma flotante?

9.23. Exprese en formato de coma flotante IEEE de 32-bits los siguientes números:

(a) -5 (c) $-1{,}5$ (e) 1/16

(b) -6 (d) 384 (f) $-1/32$

9.24. Los siguientes números emplean el formato en coma flotante IEEE de 32 bits. ¿Cuáles son sus valores decimales equivalentes?

(a) 1 10000011 11000000000000000000000

(b) 0 01111110 10100000000000000000000

(c) 0 10000000 00000000000000000000000

9.25. Considere un formato en coma flotante del IEEE, pero reducido a 7 bits, con 3 bits para el exponente y 3 bits para la parte significativa. Liste sus 127 valores.

9.26. Exprese los siguientes números en el formato de coma flotante de 32 bits de IBM, que utiliza un exponente de 7 bits con una de base implícita de 16, y un sesgo del exponente de 64 (40 en hexadecimal).

Un número en coma flotante normalizado requiere que el dígito más a la izquierda sea distinto de cero; la coma implícita de la base está a la izquierda de dicho dígito.

(a) 1,0 (b) 0,5	(c) 1/64 (d) 0,0	(e) $-15{,}0$ (f) $5{,}4 \times 10^{-79}$	(g) $7{,}2 \times 10^{75}$ (h) 65535

9.27. Suponga que el número hexadecimal 5BCA000 está en el formato de coma flotante de IBM. ¿Cuál es su valor en decimal?

9.28. ¿Cuál sería el valor de sesgo para:

(a) Un exponente de base 2 (B=2) en un campo de 6 bits?
(b) Un exponente de base 8 (B=8) en un campo de 7 bits?

9.29. Dibuje una representación de la recta real similar a la Figura 9.19b para el formato de coma flotante de la Figura 9.21b.

9.30. Considere un formato de coma flotante con 8 bits para el exponente sesgado y 23 bits para la parte significativa. Obtenga los patrones de bits de los siguientes números expresados con dicho formato:
(a) -720 (b) 0,645

9.31. Los libros mencionan que un formato de 32 bits puede representar como máximo 2^{32} números diferentes. ¿Cuántos números diferentes pueden representarse en el formato de 32 bits del IEEE? Explique la respuesta.

9.32. Cualquier representación en coma flotante utilizada en computadores representa con exactitud solo ciertos números, todos los demás deben aproximarse. Si *A'* es el valor almacenado del valor real *A*, el error relativo, *r*, se expresa como:

$$r = \frac{A - A'}{A}$$

Represente la cantidad decimal +0,4 en el siguiente formato de coma flotante: base: 2; exponente: sesgado, 4 bits; parte significativa: 7 bits. ¿Cuál es el error relativo?

9.33. Si $A = 1{,}427$, encuentre el error relativo si *A* es truncado a 1,42, y si es redondeado a 1,43.

9.34. Cuando la gente habla de la imprecisión de la aritmética en coma flotante, normalmente asocia los errores a la cancelación que tiene lugar al restar cantidades muy próximas entre sí. Pero cuando *X* e *Y* son aproximadamente iguales, la diferencia $X - Y$ se obtiene con exactitud, sin error. ¿Qué es lo que quiere decir exactamente la gente?

9.35. Los valores numéricos *A* y *B* se almacenan en un computador como los aproximados *A'* y *B'*. Despreciando cualesquiera errores de truncamiento o de redondeo posteriores, pruebe que el error relativo del producto es aproximadamente la suma de los errores relativos de los factores.

9.36 Uno de los errores más serios en los cálculos con computadores se produce al restar dos números casi iguales. Considere $A = 0{,}22288$ y $B = 0{,}22211$. El computador trunca todos los valores a 4 dígitos decimales. Por tanto $A' = 0{,}2228$ y $B' = 0{,}2221$.

(a) ¿Cuáles son los errores relativos de *A'* y *B'*?
(b) ¿Cuál es el error relativo de $C' = A' - B'$?

9.37. Para tener una noción más clara sobre los efectos de la denormalización y del desbordamiento a cero gradual, considere un sistema decimal que disponga de seis dígitos decimales para la parte significativa y que el número normalizado más pequeño sea 10^{-99}. Un número normalizado tiene un dígito decimal distinto de cero a la izquierda del punto decimal. Realice los siguientes cálculos y normalice los resultados. Comente los resultados.

(a) $(2{,}50000 \times 10^{-60}) \times (3{,}50000 \times 10^{-43})$
(b) $(2{,}50000 \times 10^{-60}) \times (3{,}50000 \times 10^{-60})$
(c) $(5{,}67834 \times 10^{-97}) - (5{,}67812 \times 10^{-97})$

9.38. Muestre cómo se realizan las siguientes sumas en coma flotante (en las que las partes significativas se truncan a cuatro dígitos decimales). Indique los resultados en forma normalizada.

(a) $0{,}566 \times 10^2 \times 7{,}777 \times 10^2$ (b) $3{,}344 \times 101 + 8{,}877 \times 10^{-2}$

9.39. Muestre cómo se realizan las siguientes restas en coma flotante (en donde las partes significativas se truncan a cuatro dígitos decimales). Indique los resultados en forma normalizada.

(a) $7{,}744 \times 10^{-3} - 6{,}666 \times 10^{-3}$ (b) $8{,}844 \times 10^{-3} - 2{,}233 \times 10^{-1}$

9.40. Muestre cómo se realizan los siguientes cálculos en coma flotante (en donde las partes significativas se truncan a cuatro dígitos decimales). Indique los resultados en forma normalizada.

(a) $(2{,}255 \times 10^1) \times (1{,}234 \times 10^0)$ (b) $(8{,}833 \times 10^2) \div (5{,}555 \times 10^4)$

CAPÍTULO 10

Repertorios de instrucciones: características y funciones

PUNTOS CLAVE

- Los elementos esenciales de las instrucciones de los computadores son el código de operación, que especifica la operación a realizar; las referencias a operandos origen y destino, que especifican la ubicación de las entradas y salidas para la operación; y la referencia a la siguiente instrucción, que usualmente está implícita.
- Los códigos de operación especifican las operaciones dentro de de las siguientes categorías generales: operaciones aritméticas y lógicas, transferencia de datos entre dos registros, entre registros y memoria, o entre dos posiciones de memoria; entrada/salida (E/S); y control.
- Las referencias a operandos especifican registros o posiciones de memoria de datos de operandos. Los datos pueden ser de diversos tipos: direcciones, números, caracteres o datos lógicos.
- Una característica arquitectural común de los procesadores es la utilización de una pila, que puede estar visible o no al programador. Las pilas se emplean para gestionar las llamadas y retornos de procedimientos, y pueden contemplarse como una forma alternativa de direccionar memoria. Las operaciones básicas con la pila son «PUSH» («introducir»), «POP» («extraer»), y operaciones con una o dos posiciones de la cabecera de la pila. Las pilas normalmente se implementan de manera que crecen de direcciones más altas hacia más bajas.
- Los procesadores pueden clasificarse como extremo mayor (*big endian*), extremo menor (*little-endian*), y ambos extremos (*bi-endian*). Un dato numérico multi-byte que se almacena con el byte más significativo en la dirección numérica más baja, se memoriza en la forma primero el extremo mayor; si se memoriza con el byte más significativo en la dirección más alta, lo hace en la forma primero el extremo menor. Un procesador ambos extremos puede manejar ambos estilos de memorización.

Gran parte de lo tratado en este libro no es fácilmente visible para el usuario o programador de un computador. Si un programador está usando un lenguaje de alto nivel, como el Pascal o el Ada, muy poco de la arquitectura de la máquina está visible.

Un punto de encuentro en que el diseñador del computador y el programador pueden ver la misma máquina es el repertorio de instrucciones. Desde el punto de vista del diseñador, el conjunto de instrucciones máquina constituye la especificación o requisitos funcionales del procesador: implementar el procesador es una tarea que, en buena parte, implica implementar el repertorio de instrucciones máquina. Desde el punto de vista del usuario, quien elige programar en lenguaje máquina (realmente en lenguaje ensamblador; *véase* Sección 10.6) se hace consciente de la estructura de registros y de memoria, de los tipos de datos que acepta directamente la máquina y del funcionamiento de la ALU.

La descripción del repertorio de instrucciones máquina de un computador es un paso más hacia la explicación del procesador del computador. De acuerdo con esto, dedicaremos este capítulo y el siguiente a las instrucciones máquina.

10.1. CARACTERISTÍCAS DE LAS INSTRUCCIONES MÁQUINA

El funcionamiento del procesador está determinado por las instrucciones que ejecuta. Estas instrucciones se denominan *instrucciones máquina* o *instrucciones del computador*. Al conjunto de instrucciones distintas que puede ejecutar el procesador se denomina *repertorio de instrucciones* del procesador.

ELEMENTOS DE UNA INSTRUCCIÓN MÁQUINA

Cada instrucción debe contener la información que necesita el procesador para su ejecución. La Figura 10.1, que es una repetición de la Figura 3.6, muestra los pasos involucrados en la ejecución de instrucciones, e implícitamente define los elementos constitutivos de una instrucción máquina. Dichos elementos son:

- **Código de operación:** especifica la operación a realizar (suma, E/S, etc.). La operación se indica mediante un código binario denominado código de operación o, abreviadamente, **codop**.
- **Referencia a operandos fuente u origen:** la operación puede implicar a uno o más operandos origen, es decir operandos que son entradas para la instrucción.
- **Referencia al operando de destino o resultado:** la operación puede producir un resultado.
- **Referencia a la siguiente instrucción:** dice al procesador de dónde captar la siguiente instrucción tras completarse la ejecución de la instrucción actual.

La siguiente instrucción a captar está en memoria principal o, en el caso de un sistema de memoria virtual, bien en memoria principal o en memoria secundaria (disco). En la mayoría de los casos, la siguiente instrucción a captar sigue inmediatamente a la instrucción en ejecución. En tales casos no

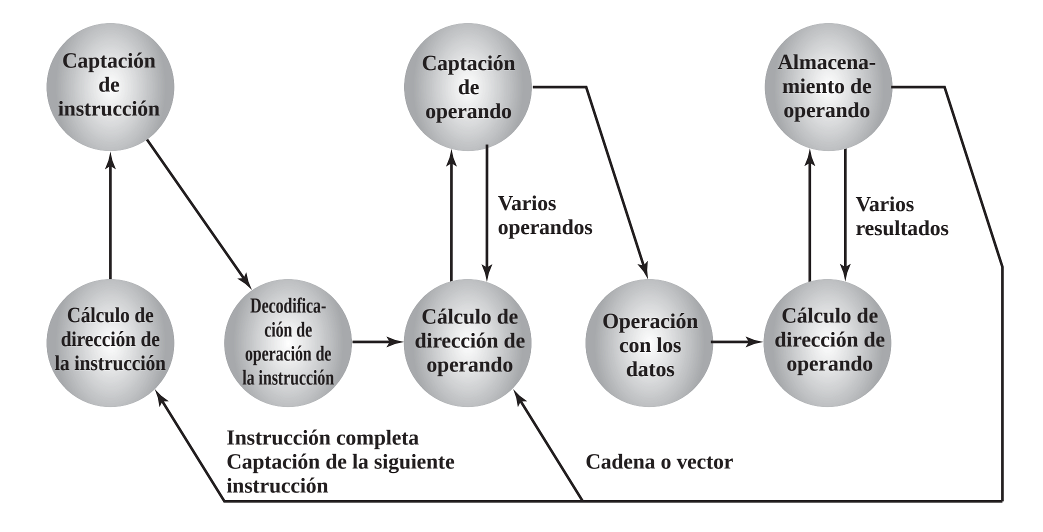


Figura 10.1. Diagrama de estados de un ciclo de instrucción.

hay referencia explícita a la siguiente instrucción. Cuando sea necesaria una referencia explícita, debe suministrarse la dirección de memoria principal o de memoria virtual. La forma en que se da dicha dirección se discute en el Capítulo 11.

Los operandos origen y destino pueden estar en alguna de las tres áreas siguientes:

- **Memoria principal o virtual:** como en las referencias a instrucciones siguientes, debe indicarse la dirección de memoria principal o de memoria virtual.
- **Registro del procesador:** salvo raras excepciones, un procesador contiene uno o más registros que pueden ser referenciados por instrucciones máquina. Si solo existe un registro, la referencia a él puede ser implícita. Si existe más de uno, cada registro tendrá asignado un número único y la instrucción debe contener el número del registro deseado.
- **Dispositivo de E/S:** la instrucción debe especificar el módulo y dispositivo de E/S para la operación. En el caso de E/S asignadas en memoria, se dará otra dirección de memoria principal o virtual.

REPRESENTACIÓN DE LAS INSTRUCCIONES

Dentro del computador, cada instrucción se representa por una secuencia de bits. La instrucción está dividida en campos correspondientes a los elementos constitutivos de la misma. La Figura 10.2 muestra un ejemplo sencillo de formato de instrucción. Otro ejemplo, el formato de instrucciones del IAS, se mostró en la Figura 2.2. En la mayoría de los repertorios de instrucciones se emplea más de un formato. Durante su ejecución, la instrucción se escribe en un registro de instrucción (IR) del procesador. El procesador debe ser capaz de extraer los datos de los distintos campos de la instrucción para realizar la operación requerida.

Es difícil, tanto para los programadores como para los lectores de un libro de texto, manejar las representaciones binarias de las instrucciones máquina. Por ello, es una práctica común utilizar *representaciones simbólicas* de las instrucciones máquina. Un ejemplo se dio en la Tabla 2.1 para el repertorio de instrucciones del IAS.

Los codops se representan mediante abreviaturas, denominadas *nemotécnicos*, que indican la operación en cuestión. Ejemplos usuales son:

ADD	Sumar
SUB	Restar
MPY	Multiplicar
DIV	Dividir
LOAD	Cargar datos de memoria
STOR	Almacenar datos en memoria (memorizar)

Los operandos también suelen representarse simbólicamente. Por ejemplo, la instrucción

ADD R,Y

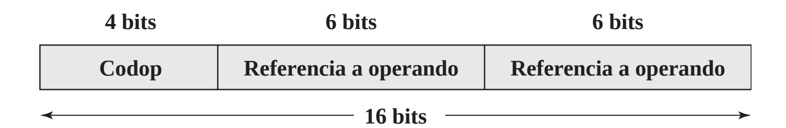


Figura 10.2. Un formato de instrucciones sencillo.

puede significar sumar el valor contenido en la posición de datos Y al contenido del registro R. En este ejemplo, Y hace referencia a la dirección de una posición de memoria, y R a un registro particular. Observe que la operación se realiza con el contenido de la posición, no con su dirección.

Es posible pues escribir un programa en lenguaje máquina de forma simbólica. Cada codop simbólico tiene una representación binaria fija, y el programador especifica la posición de cada operando simbólico. Por ejemplo, el programador podría comenzar con una lista de definiciones:

$$X = 513$$

$$Y = 514$$

y así sucesivamente. Un sencillo programa aceptaría como entrada esta información simbólica, convertiría los codops y referencias a operandos a forma binaria y construiría las instrucciones máquina binarias.

Es raro encontrar ya programadores en lenguaje máquina. La mayoría de los programas actuales se escriben en un lenguaje de alto nivel o, en ausencia del mismo, en lenguaje ensamblador, sobre el que trataremos al final de este capítulo. No obstante, el lenguaje máquina simbólico sigue siendo útil para describir las instrucciones máquina, y con ese fin lo utilizaremos.

TIPOS DE INSTRUCCIONES

Considere una instrucción de alto nivel tal y como se expresaría en un lenguaje como el BASIC o el FORTRAN. Por ejemplo,

$$X = X + Y$$

Esta sentencia ordena al computador sumar los valores almacenados en X y en Y, y poner el resultado en X. ¿Cómo se podría realizar lo mismo con instrucciones máquina? Supongamos que las variables X e Y corresponden a las posiciones 513 y 514. Considerando un repertorio simple de instrucciones máquina, la operación podría llevarse a cabo con tres instrucciones:

1. Cargar un registro con el contenido de la posición de memoria 513.
2. Sumar al registro el contenido de la posición de memoria 514.
3. Memorizar el contenido del registro en la posición de memoria 513.

Como se observa, una sola instrucción BASIC puede necesitar de tres instrucciones máquina. Este es un caso típico de relación entre un lenguaje de alto nivel y un lenguaje máquina. Un lenguaje

de alto nivel expresa las operaciones de forma algebraica concisa, utilizando variables. Un lenguaje máquina expresa las operaciones de una manera elemental, implicando operaciones de transferencia de datos a o desde registros.

Con el sencillo ejemplo anterior como guía, consideremos qué tipos de instrucciones deben incluirse en un computador real. Debiera tener un conjunto de instrucciones que permitieran al usuario formular cualquier tarea de procesamiento de datos. Otra forma de verlo sería considerar las posibilidades de un lenguaje de programación de alto nivel. Cualquier programa escrito en alto nivel debe traducirse a lenguaje máquina para ser ejecutado. Por tanto, el repertorio de instrucciones máquina debe ser suficientemente amplio como para expresar cualquiera de las instrucciones de un lenguaje de alto nivel. Teniendo esto presente, los tipos de instrucciones se pueden clasificar de la siguiente manera:

- De **procesamiento de datos:** instrucciones aritméticas y lógicas
- De **almacenamiento de datos:** instrucciones de memoria.
- De **transferencia de datos:** instrucciones de E/S.
- De **control:** instrucciones de comprobación y de bifurcación.

Las instrucciones *aritméticas* proporcionan capacidad computacional para procesar datos numéricos. Las instrucciones *lógicas* (booleanas) operan con los bits de una palabra en lugar de considerarlos como números, proporcionando por tanto capacidad para el procesamiento de cualquier otro tipo de datos que el usuario quiera emplear. Estas operaciones se realizan principalmente con datos en registros del procesador. Por lo tanto, debe haber instrucciones de *memoria* para transferir los datos entre la memoria y los registros. Las instrucciones de E/S se necesitan para transferir programas y datos a memoria y devolver resultados de los cálculos al usuario. Las instrucciones de *comprobación* o test se emplean para comprobar el valor de una palabra de datos o el estado de un cálculo. Las de *bifurcación* se usan entonces para bifurcar a diferentes conjuntos de instrucciones dependiendo de la decisión tomada.

Examinaremos los distintos tipos de instrucciones con mayor detalle más adelante, en este mismo capítulo.

NÚMERO DE DIRECCIONES

Una de las formas tradicionales de describir la arquitectura de un procesador es en términos del número de direcciones contenidas en cada instrucción. Esta dimensión se va haciendo menos significativa a medida que aumenta la complejidad del diseño del procesador. A pesar de ello, merece la pena extenderse y analizar dicha distinción.

¿Cuál es el número máximo de direcciones que serían necesarias en una instrucción? Evidentemente, las instrucciones aritméticas y lógicas son las que requieren más operandos. Prácticamente todas las operaciones aritméticas y lógicas son o bien unarias (un operando) o binarias (dos operandos). Así pues, necesitaríamos un máximo de dos direcciones para referenciar operandos. El resultado de una operación debe almacenarse, lo que sugiere una tercera dirección. Finalmente, tras completar una instrucción debe captarse la siguiente, y su dirección es pues necesaria.

El razonamiento anterior sugiere como plausible que una instrucción incluyera cuatro referencias a direcciones: dos operandos, un resultado y la dirección de la instrucción siguiente. En la práctica es muy raro encontrar instrucciones que contengan cuatro direcciones. La mayoría de las instrucciones tienen una, dos o tres direcciones, estando implícita la dirección de la instrucción siguiente (obtenida a partir del contador de programa).

La Figura 10.3 compara instrucciones típicas de una, dos y tres direcciones, que podrían utilizarse para calcular Y = (A − B) / [(C + D (E)]. Con tres direcciones, cada instrucción especifica dos posiciones de operandos y la posición del resultado. Dado que no queremos alterar el valor de ninguna posición de operando, se utiliza una posición temporal, T, para almacenar resultados intermedios. Observe que hay cuatro instrucciones y que la expresión original tenía cinco operandos.

Las instrucciones con tres direcciones no son comunes ya que requieren formatos relativamente largos para albergar las tres referencias. Con instrucciones de dos direcciones, para operaciones binarias una de las direcciones debe hacer el servicio doble de uno de los operandos y del resultado. Así pues, la instrucción «SUB Y», B realiza el cálculo Y − B y guarda el resultado en Y. El formato de dos direcciones reduce el espacio necesario pero resulta algo engorroso. Para evitar que se altere el valor de un operando se utiliza una instrucción «MOVE» para transferir uno de los valores a una posición temporal o de resultados, antes de realizar el cálculo. Nuestro programa ejemplo se amplía a seis instrucciones.

La instrucción de una sola dirección es aún más simple. Para que funcione, una segunda dirección debe estar implícita. Esto fue lo usual en las primeras máquinas, en las que la dirección implícita era un registro del procesador conocido como **acumulador** (AC). El acumulador contiene uno de los operandos y se emplea para almacenar el resultado. En nuestro ejemplo se necesitarían ocho instrucciones para realizar la tarea.

Instrucción		Comentario
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

(a) Instrucciones de tres direcciones

Instrucción		Comentario
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T \times E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

(b) Instrucciones de dos direcciones

Instrucción		Comentario
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC \times E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

(c) Instrucciones de una dirección

Figura 10.3. Programas para calcular $Y = \dfrac{(A - B)}{(C + D \times E)}$.

Es de hecho posible arreglárselas con cero direcciones para algunas de instrucciones. Las instrucciones con cero direcciones son aplicables a una organización especial de memoria, denominada *pila* («stack»). Una pila es un conjunto de posiciones del tipo *last-in-first-out* (el primero en entrar es primero en salir). La pila está en una posición conocida y, a menudo, al menos los dos elementos de su cabecera están en registros del procesador. Así pues las instrucciones con cero direcciones referenciarían dichos elementos de la cabecera. Las memorias pila se describen en el Apéndice 10A, y su uso se trata también en el Capítulo 11.

La Tabla 10.1 resume posibles interpretaciones a considerar para las instrucciones de cero, una dos, o tres direcciones. En todos los casos se supone que la dirección de la siguiente instrucción está implícita, y que se va a realizar una operación con dos operandos origen y un resultado.

El número de direcciones por instrucción es una decisión básica de diseño. Menos direcciones por instrucción significa instrucciones más primarias, lo que requiere un procesador menos complejo. También da lugar a instrucciones más cortas. Por otra parte, los programas contienen más instrucciones, lo que normalmente supone mayor tiempo de ejecución y programas más largos y complejos. Hay también un umbral importante entre instrucciones de una y de múltiples direcciones. Con instrucciones de una sola dirección, el programador tiene generalmente a su disposición solo un registro de uso general, el acumulador. Con instrucciones de múltiples direcciones suele disponerse de múltiples registros de uso general. Esto permite que algunas operaciones se realicen solo con registros. Ya que los accesos a registros son más rápidos que a memoria se acelera la ejecución. Por razones de flexibilidad y facilidad para utilizar varios registros, la mayoría de las máquinas contemporáneas emplean una combinación de instrucciones de dos y de tres direcciones.

Las decisiones de diseño asociadas con la elección del número de direcciones por instrucción son complicadas debido a otros factores. Un aspecto a considerar es si una dirección hace referencia a una posición de memoria o a un registro. Ya que hay menos registros, se necesitan menos bits para referenciarlos. Además, como veremos en el capítulo siguiente, una máquina puede permitir diversos modos de direccionamiento, y la especificación del modo consume uno o más bits. El resultado es que la mayoría de los diseños de procesadores hacen uso de varios formatos de instrucciones.

Tabla 10.1. Utilización de las direcciones de las instrucciones (instrucciones sin bifurcación).

Número de direcciones	Representación simbólica	Intrepretación
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T − 1) OP T

AC = Acumulador.
T = Cabecera de la pila.
(T − 1) = Segundo elemento de la pila.
A, B, C = Posiciones de memoria o registros.

DISEÑO DEL REPERTORIO DE INSTRUCCIONES

Uno de los aspectos más interesantes, y más analizados, del diseño de un computador es el diseño del repertorio de instrucciones del lenguaje máquina. El diseño de un repertorio de instrucciones es muy complejo ya que afecta a muchos aspectos del computador. El repertorio de instrucciones define muchas de las funciones realizadas por el procesador, y tiene pues un efecto significativo sobre la implementación del mismo. El repertorio de instrucciones es el medio que tiene el programador para controlar el procesador. En consecuencia deben considerarse las necesidades del programador a la hora de diseñar el repertorio de instrucciones.

Puede sorprender saber que algunos de los aspectos más básicos relativos al diseño de repertorios de instrucciones siguen siendo temas de controversia. Realmente, en los últimos años, el grado de desacuerdo ha aumentado. Los más importantes entre dichos aspectos fundamentales de diseño son:

- El **repertorio de operaciones:** cuántas y qué operaciones considerar, y cuán complejas deben ser.
- Los **tipos de datos:** los distintos tipos de datos con los que se efectúan operaciones.
- Los **formatos de instrucciones:** longitud de la instrucción (en bits), número de direcciones, tamaño de los distintos campos, etc.
- Los **registros:** número de registros del procesador que pueden ser referenciados por las instrucciones, y su uso.
- El **direccionamiento:** el modo o modos de direccionamiento mediante los cuales puede especificarse la dirección de un operando.

Estos aspectos están fuertemente interrelacionados y deben considerarse conjuntamente en el diseño de un repertorio de instrucciones. Este libro, por supuesto, debe considerarlos en secuencia, pero se intentará mostrar las relaciones entre ellos.

Dada la importancia del tema, un bloque amplio de la Parte Tres del libro se dedica al diseño del repertorio de instrucciones. A continuación de esta sección de repaso, el capítulo examina los tipos de datos y repertorios de operaciones. El Capítulo 11 trata los modos de direccionamiento (considerando también los registros) y los formatos de instrucciones. El Capítulo 13 se dedica al estudio del «computador de conjunto reducido de instrucciones» (RISC, *Reduced Instruction Set Computer*). La arquitectura RISC cuestiona muchas de las decisiones sobre repertorios de instrucciones tradicionalmente adoptadas en computadores comerciales.

10.2. TIPOS DE OPERANDOS

Las instrucciones máquina operan con datos. Las categorías generales más importantes de datos son:

- Direcciones.
- Números.
- Caracteres.
- Datos lógicos.

Cuando tratemos los modos de direccionamiento en el Capítulo 11, veremos que las direcciones son de hecho un tipo de datos. En muchos casos debe realizarse algún cálculo sobre la referencia a un operando de una instrucción a fin de determinar la dirección de memoria principal o virtual. En este contexto, las direcciones pueden considerarse como números enteros sin signo.

Otros tipos de datos comunes son los números, los caracteres y los datos lógicos; y cada uno de ellos se analiza brevemente en esta sección. Aparte de estos, en algunas máquinas se utilizan tipos o estructuras de datos específicos. Por ejemplo, puede haber operaciones máquina que trabajan directamente con listas o cadenas de caracteres.

NÚMEROS

Todos los lenguajes máquina incluyen tipos de datos numéricos. Incluso en el procesamiento de datos no numéricos se necesitan números que actúen como contadores, longitudes de campos, etc. Una distinción importante entre los números utilizados en las matemáticas ordinarias y los almacenados en un computador es que estos últimos están limitados. Esto es cierto en dos sentidos. En primer lugar, hay un límite para la magnitud de los números representables en una máquina y, en segundo lugar, en el caso de números en coma flotante, su precisión está limitada. Por tanto, el programador debe ser consciente de las consecuencias del redondeo, el desbordamiento o el desbordamiento a cero.

En los computadores son usuales tres tipos de datos numéricos:

- Enteros o en coma fija.
- En coma flotante.
- En decimal.

Los dos primeros se vieron con detalle en el Capítulo 9. Falta comentar algo sobre los números decimales.

Aunque todas las operaciones internas del computador son en esencia binarias, los usuarios del sistema utilizamos números decimales. Es necesario, pues, convertir de decimal a binario las entradas y de binario a decimal las salidas. Para aplicaciones en las que hay muchas entradas/salidas frente a pocos cálculos, y cálculos comparativamente simples, es preferible memorizar y operar con los números directamente en su forma decimal. La representación más común para ello es la de **decimal empaquetado** [1].

En decimal empaquetado, cada dígito decimal se representa mediante un código de cuatro bits, según la manera habitual. Es decir, 0 = 0000, 1 = 0001, ..., 8 = 0100, y 9 = 1001. Observe que resulta un código bastante ineficiente ya que solo se emplean diez de las 16 posibles combinaciones de cuatro bits. Para construir números, se van encadenando códigos de 4 bits, normalmente formando múltiplos de 8 bits. Así, el código de 246 es 0000001001000110. Este código es claramente menos compacto que la representación binaria directa, pero evita la necesidad de conversiones. Los números negativos pueden representarse incluyendo un dígito de signo de cuatro bits bien a la izquierda o a la

[1] Los libros de texto la suelen denominar «decimal codificado en binario» (BCD). Hablando con propiedad, BCD se refiere a la codificación de cada dígito decimal mediante una secuencia única de cuatro bits. Decimal empaquetado hace referencia al almacenamiento de dígitos codificados en BCD empleando un byte para cada pareja de dígitos.

derecha de la cadena de dígitos decimales empaquetados. Por ejemplo, el código 1111 podría representar el signo menos.

Muchas máquinas tienen instrucciones aritméticas destinadas a operar directamente con números en decimal empaquetado. Los algoritmos correspondientes son bastante similares a los descritos en la Sección 9.3 pero deben tener en cuenta la operación de acarreo decimal.

CARACTERES

Una forma bastante común de datos es el texto o secuencias de caracteres. Aunque la información textual sea más conveniente para las personas, no puede ser almacenada o transmitida fácilmente en forma de caracteres por los sistemas de comunicación y de procesamiento de datos. Tales sistemas están diseñados para datos binarios. Por lo tanto, se han ideado diversos códigos que permiten representar caracteres mediante secuencias de bits. Tal vez el primer ejemplo fue el código Morse. Hoy en día, el código de caracteres más utilizado es el alfabeto de referencia internacional (IRA, *International Reference Alphabet*), conocido en los Estados Unidos como ASCII (*American Standard Code for Information Interchange*): código estándar americano para intercambio de información (*véase* Tabla 7.1). Cada carácter es representado en este código por un patrón distinto de 7 bits; pueden representarse por tanto 128 caracteres diferentes. Este número es mayor que el necesario para representar los caracteres impresos, utilizándose algunos de los patrones para representar caracteres de *control*. Algunos de estos se emplean para controlar la impresión de caracteres en una página. Otros caracteres de control están dedicados para procedimientos de comunicación. Los caracteres codificados en ASCII se memorizan y transmiten utilizando casi siempre ocho bits por carácter. El octavo bit puede fijarse a cero o utilizarse como bit de paridad para la detección de errores. En el segundo caso, el valor del bit se pone de manera que el número total de unos en cada octeto sea o siempre impar (paridad impar) o siempre par (paridad par).

Observe en la Tabla 7.1 que, con el ASCII, los patrones de bits 011XXXX representan los dígitos 0 a 9 mediante sus valores binarios equivalentes: 0000 a 1001, en los cuatro bits de la derecha. Esta codificación coincide con la empleada en decimal empaquetado, facilitándose así la conversión entre ASCII de siete bits y decimal empaquetado de cuatro bits.

Otro código utilizado para caracteres es el código de intercambio decimal codificado en binario ampliado (EBCDIC, *Extended Binary Coded Decimal Interchange Code*). Este código, de ocho bits, se emplea en los grandes computadores de IBM. Como el ASCII, el EBCDIC es compatible con el decimal empaquetado. En el caso del EBCDIC, los códigos 11110000 a 11111001 representan los dígitos 0 a 9.

DATOS LÓGICOS

Normalmente, cada palabra o cualquier otra unidad direccionable (byte, media palabra, etc.) es tratada como una unidad de datos individual. Sin embargo, a veces es útil considerar una unidad de n bits como n elementos o datos de un bit, donde cada elemento tiene un valor 1 o 0. Cuando los datos son vistos de esta manera, se consideran datos *lógicos*.

Esta representación orientada a bits tiene dos ventajas. La primera es que a veces puede interesarnos almacenar una matriz de elementos binarios o booleanos, en la que cada elemento pueda tomar solo los valores 1 (verdadero) ó 0 (falso). Con datos lógicos, la memoria puede ser entonces utilizada

más eficientemente. En segundo lugar, hay ocasiones en las que queremos manipular bits individuales de un dato. Por ejemplo, cuando se implementan operaciones de coma flotante en software, necesitamos desplazar bits significativos en algunas operaciones. Otro ejemplo es al convertir de ASCII a decimal empaquetado, donde se necesita extraer los cuatro bits de la derecha de cada byte.

Obsérvese que, en los ejemplos precedentes, los mismos datos son unas veces tratados como datos lógicos y otras como numéricos o como texto. El «tipo» de una unidad de datos viene determinado por la operación que se esté realizando con ella. Aunque esto no sea lo normal con los lenguajes de alto nivel, es casi siempre así en el caso de lenguajes máquina.

10.3. TIPOS DE DATOS EN EL PENTIUM Y EL POWERPC

TIPOS DE DATOS EN EL PENTIUM

El Pentium puede tratar tipos de datos de 8 (byte), 16 (palabra), 32 (palabra doble) y 64 (palabra cuádruple) bits de longitud. Para posibilitar una flexibilidad máxima en las estructuras de datos y una utilización eficiente de la memoria, las palabras no tienen por qué estar alineadas con las direcciones pares de memoria, ni las palabras dobles alineadas con las direcciones divisibles por cuatro, ni las cuádruples con direcciones divisibles por ocho. Sin embargo, cuando se accede a los datos a través de un bus de 32 bits, su transferencia tiene lugar en unidades de palabras dobles, empezando en direcciones divisibles por cuatro. El procesador convierte las peticiones con valores no alineados en una secuencia de peticiones adaptada a la forma de transferencia en el bus. Como en todas las máquinas Intel 80X86, el Pentium emplea el estilo extremo menor; es decir, el byte menos significativo es almacenado en la dirección más baja (*véase* el Apéndice 10B).

El byte, la palabra, la palabra doble y la cuádruple son referidas como tipos generales de datos. Además, el Pentium admite una variedad impresionante de tipos de datos específicos que son reconocidos y procesados mediante instrucciones concretas. La Tabla 10.2 resume estos tipos.

Tabla 10.2. Tipos de datos del Pentium.

Tipo de datos	Descripción
General	Posiciones de byte, de palabra (16 bits), de palabra doble (32 bits), y de palabra cuádruple (64 bits), con contenido binario arbitrario
Entero	Un valor binario con signo contenido en un byte, una palabra o una palabra doble, representado en complemento a dos.
Ordinal	Un entero sin signo contenido en un byte, una palabra o una palabra doble.
Decimal codificado en binario (BCD) desempaquetado	Representación de un dígito BCD en el rango de 0 a 9, con un dígito en cada byte.
BCD empaquetado	Representación empaquetada de dos dígitos BCD en un byte; valor en el rango de 0 a 99.

(continúa)

Tabla 10.2. Tipos de datos del Pentium *(continuación).*

Tipo de datos	Descripción
Puntero de proximidad	Una dirección efectiva de 32 bits que representa el desplazamiento dentro de un segmento. Utilizado para todos los punteros en una memoria no segmentada y para referencias dentro de un segmento en una memoria segmentada.
Campo de bits	Una secuencia contigua de bits en la que cada posición de bit se considera como unidad independiente. Una cadena de bits puede comenzar en cualquier posición de cualquier byte y puede contener hasta $2^{32} - 1$ bits.
Cadena de bytes	Una secuencia contigua de bytes, de palabras, o de palabras dobles, que contiene de 0 a $2^{32} - 1$ bytes.
Coma flotante	*Véase* la Figura 10.4.

La Figura 10.4 muestra los tipos de datos numéricos del Pentium. Los enteros con signo están en representación de complemento a dos y pueden ser de 16, 32 o 64 bits. El tipo coma flotante se refiere realmente a un conjunto de tipos utilizados por la unidad de coma flotante y que son procesados mediante instrucciones de coma flotante. Las tres representaciones en coma flotante se ajustan al estándar IEEE 754.

TIPOS DE DATOS EN EL POWERPC

El PowerPC puede manejar tipos de datos de 8 (byte), 16 (media palabra), 32 (palabra) y 64 (palabra doble) bits de longitud. Algunas instrucciones requieren que los operandos de memoria estén alineados con una frontera de 32 bits. Sin embargo, en general no es necesario el alineamiento. Una característica interesante del PowerPC es que puede utilizar bien el estilo extremo menoro el extremo mayor; es decir, el byte menos significativo puede estar almacenado bien en la dirección más baja o en la más alta (*véase* el Apéndice 10B).

El byte, la media palabra, la palabra, y la palabra doble son tipos de datos generales. El procesador interpreta el contenido de un cierto elemento de datos dependiendo de la instrucción. El procesador de coma fija reconoce los siguientes tipos de datos:

- **Byte sin signo:** puede utilizarse para operaciones lógicas o para aritméticas con enteros. Se carga de memoria en un registro general completando con ceros hacia la izquierda hasta la longitud total del registro.
- **Media palabra sin signo:** como el byte sin signo, pero para cantidades de 16 bits.
- **Media palabra con signo:** utilizado para operaciones aritméticas; cargado en memoria completando el signo hacia la izquierda hasta la longitud total del registro (es decir, se replica el bit de signo en todas las posiciones vacantes).
- **Palabra sin signo:** utilizado para operaciones lógicas y como puntero de direcciones.
- **Palabra con signo:** utilizado para operaciones aritméticas.

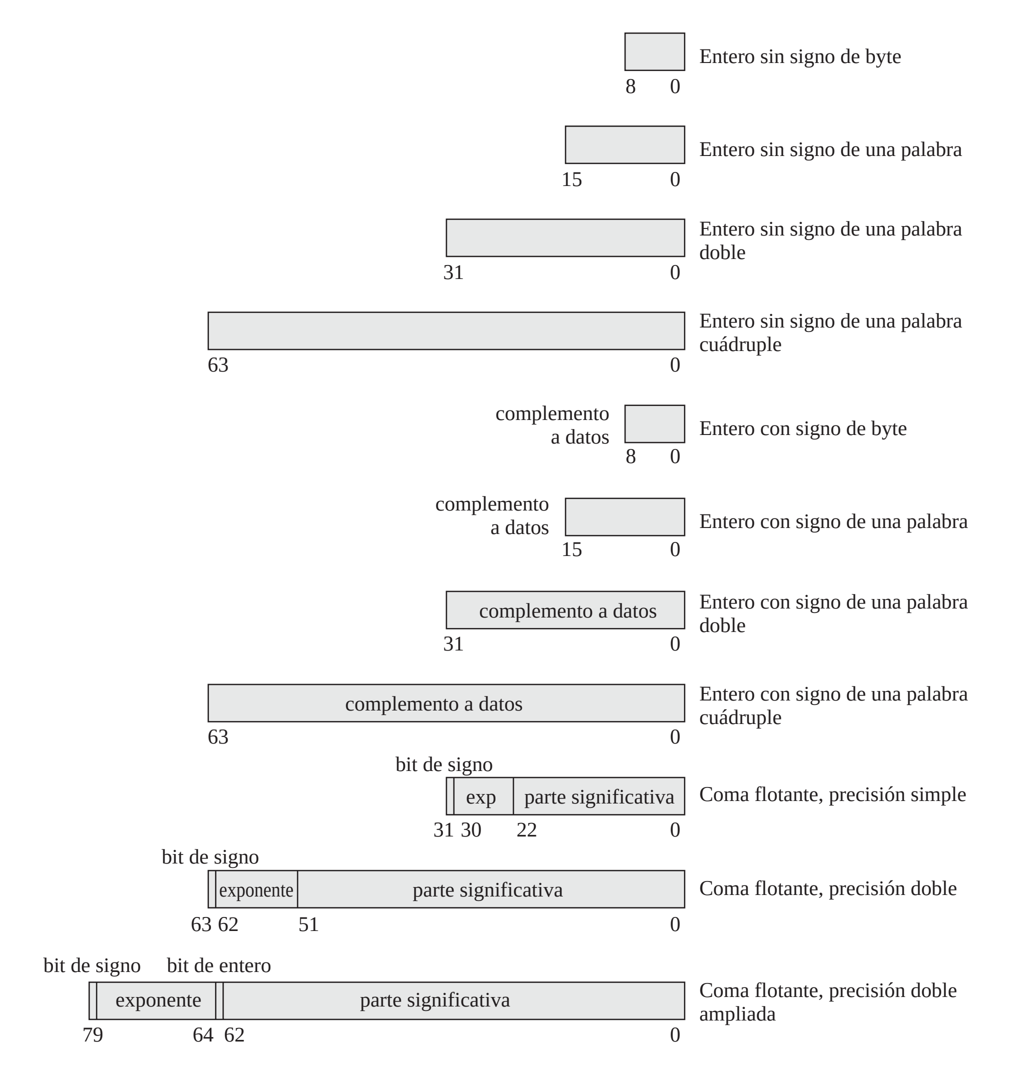


Figura 10.4. Formatos de datos numéricos en el Pentium.

- **Palabra doble sin signo:** utilizado como puntero de direcciones.
- **Cadena de bytes:** desde 0 hasta 128 bytes de longitud.

Además, el PowerPC admite los tipos de datos en coma flotante de precisión simple y doble definidos en el estándar IEEE 754.

10.4. TIPOS DE OPERACIONES

El número de códigos de operación (codops) diferentes varía ampliamente de una máquina a otra. Sin embargo, en todas las máquinas podemos encontrar los mismos tipos generales de operaciones. Una clasificación típica y útil es la siguiente:

- Transferencias de datos.
- Aritméticas.
- Lógicas.
- De conversión.
- De E/S.
- De control del sistema.
- De control de flujo.

La Tabla 10.3 (basada en [HAYE88]) enumera tipos de instrucciones comunes de cada clase. Esta sección proporciona una revisión somera de los distintos tipos de operaciones, junto con una breve discusión sobre las acciones que realiza el procesador para ejecutar cada tipo particular de operación (resumidas en la Tabla 10.4). Este último punto se analiza con más detalle en el Capítulo 12.

Tabla 10.3. Operaciones usuales de repertorios de instrucciones.

Tipo	Nombre de la operación	Descripción
Transferencias de datos	*Move* (transferir)	Transfiere una palabra o un bloque desde un origen a un destino
	Store (memorizar)	Transfiere una palabra desde el procesador a memoria
	Load (cargar o captar)	Transfiere una palabra desde memoria al procesador
	Exchange (intercambiar)	Intercambia los contenidos del origen y el destino
	Clear (reiniciar o poner a 0)	Transfiere una palabra de ceros al destino
	Set (poner a 1)	Transfiere una palabra de unos al destino
	Push (introducir en la pila, «apilar»)	Transfiere una palabra desde un origen a la cabecera de la pila
	Pop (extraer de la pila, «desapilar»)	Transfiere una palabra desde la cabecera de la pila a un destino
Aritméticas	*Add* (sumar)	Calcula la suma de dos operandos
	Substract (restar)	Calcula la diferencia de dos operandos
	Multiply (multiplicar)	Calcula el producto de dos operandos
	Divide (dividir)	Calcula el cociente de dos operandos
	Absolute (valor absoluto)	Sustituye el operando por su valor absoluto
	Negate (opuesto)	Cambia el signo del operando
	Increment (incrementar)	Suma 1 al operando
	Decrement (decrementar)	Resta 1 del operando

(continúa)

Tabla 10.3. Operaciones usuales de repertorios de instrucciones *(continuación).*

Tipo	Nombre de la operación	Descripción
Lógicas	*AND* (Producto lógico, Y)	Realiza la operación lógica Y (AND)
	OR (Suma lógica, O)	Realiza la operación lógica O (OR)
	NOT (complemento)	Realiza el complemento (NOT) bit a bit del dato
	Exclusive-OR (OR-exclusiva)	Realiza la operación lógica O-Exlusiva (XOR)
	Test (comprobar)	Comprueba la condición especificada; fija los indicadores (*flags*) en función del resultado
	Compare (comparar)	Realiza la comparación lógica o aritmética de dos o más operandos; fija los indicadores (*flags*) en función del resultado
	Set control variables (fijar variables de control)	Instrucciones que fijan controles para protección, gestión de interrupciones, control del temporizador, etc.
	Shift (desplazamiento)	Desplaza el operando a la izquierda (derecha), introduciendo valores constantes por el otro extremo
	Rotate (rotar)	Desplaza el operando a la izquierda (derecha) de forma cíclica
Control de flujo	*Jump* (bifurcación o salto)	Ruptura incondicional de flujo; carga el PC con la dirección especificada
	Jump condicional (salto condicional)	Comprueba la condición especificada; dependiendo de la condición, o carga el PC con la dirección indicada, o no hace nada
	Jump to subrutina (Llamada a subrutina)	Guarda la información de control del programa en una posición conocida y salta a la dirección indicada
	Return (retorno)	Sustituye el contenido del PC y de otros registros por los de la posición conocida
	Execute (ejecutar)	Capta el operando de la dirección indicada y lo ejecuta como una instrucción; no modifica el PC
	Skip (salto implícito)	Incrementa el PC de manera que se salte la instrucción siguiente
	Skip conditional (salto implícito condicional)	Comprueba la condición indicada; realiza el salto implícito o no hace nada, dependiendo de la condición
	Halt (parar)	Detiene la ejecución del programa
	Wait (esperar)	Detiene la ejecución del programa; comprueba de forma repetitiva la condición especificada; reanuda la ejecución cuando se satisface la condición
	No operation (no operación)	No se ejecuta operación alguna, pero la ejecución del programa continúa

(continúa)

Tabla 10.3. Operaciones usuales de repertorios de instrucciones *(continuación).*

Tipo	Nombre de la operación	Descripción
Entrada/Salida	*Input* (entrada)	Transfiere datos desde un puerto o dispositivo de E/S al destino (memoria principal o registro del procesador)
	Output (salida)	Transfiere datos desde el origen especificado a un puerto o dispositivo de E/S
	Start I/O (iniciar E/S)	Transfiere instrucciones al procesador de E/S para iniciar operaciones de E/S
	Test I/O (comprobar E/S)	Transfiere información de estado desde el sistema de E/S al destino especificado
Conversión	*Translate* (traducir)	Traducción de los valores de una sección de memoria, basada en una tabla de correspondencia
	Convert (convertir)	Convierte el contenido de una palabra de un formato a otro (por ejemplo, de decimal empaquetado a binario)

Tabla 10.4. Acciones del procesador para varios tipos de operadores.

Transferencia de datos	Transfiere datos de una posición a otra Si se implica a la memoria: Determina la dirección de memoria Realiza la transformación de direcciones de memoria virtual a real Comprueba la caché Inicia la lectura/escritura en/de memoria
Aritmética	Puede implicar transferencias de datos antes y/o después Realiza la operación en la ALU Actualiza códigos e indicadores de condición
Lógica	Lo mismo que en una aritmética
Conversión	Similar a la aritmética y la lógica. Puede implicar a lógica especial para realizar la conversión
Control de flujo	Actualiza el contador de programa. En el caso de llamadas y retornos de subrutinas, gestiona la transferencia y enlace de parámetros
E/S	Cursa una orden a un módulo de E/S En el caso de E/S asignada en memoria, determina la dirección de memoria correspondiente

TRANSFERENCIA DE DATOS

El tipo de instrucción máquina más básico es la transferencia de datos. La instrucción de transferencia de datos debe especificar varias cosas. En primer lugar deben especificarse las posiciones de los operandos origen y destino. Cada posición podría ser de memoria, un registro o la cabecera de la pila. En segundo lugar, debe indicarse la longitud de los datos a transferir. Tercero, como en todas las instrucciones con operandos, debe especificarse el modo de direccionamiento para cada operando. Este último punto se trata en el Capítulo 11.

La elección de las instrucciones de transferencia de datos a incluir en un repertorio de instrucciones es un ejemplo del tipo de compromisos que debe adoptar un diseñador. Por ejemplo, la posición en general (memoria o registro) de un operando puede indicarse bien en la especificación del codop o del operando. La Tabla 10.5 muestra ejemplos de las instrucciones de transferencia de datos más comunes del IBM S/390. Observe cómo hay variantes para indicar la cantidad de datos a

Tabla 10.5. Ejemplos de operaciones de transferencia de datos del IBM S/390

Nemotécnico de la operación	Nombre	Número de bits transferidos	Descripción
L	Load	32	Transferencia de memoria a registro
LH	Load Halfword	16	Transferencia de memoria a registro (media palabra)
LR	Load	32	Transferencia de registro a registro
LER	Load (Short)	32	Transferencia (corta) entre dos registros de coma flotante
LE	Load (Short)	32	Transferencia (corta) de memoria a registro de coma flotante
LDR	Load (Long)	64	Transferencia (larga) entre dos registros de coma flotante
LD	Load (Long)	64	Transferencia (larga) de memoria a registro de coma flotante
ST	Store	32	Transferencia de registro a memoria
STH	Store Halfword	16	Transferencia de registro a memoria (media palabra)
STC	Store Character	8	Transferencia de registro a memoria (un carácter)
STE	Store (Short)	32	Transferencia (corta) de registro de coma flotante a memoria
STD	Store (Long)	64	Transferencia (larga) de registro de coma flotante a memoria

transferir (8, 16, 32, o 64 bits). También hay diferentes instrucciones para transferencias entre registros, de registro a memoria, y de memoria a registro. Por contra, el VAX tiene una instrucción de transferencia (MOV) con variantes según la cantidad de datos a transferir, pero que especifica como parte del operando si este está en un registro o en memoria. La aproximación considerada en el VAX es algo más fácil para el programador, que tendría que manejar menos nemotécnicos. Sin embargo, es menos compacta que la considerada en el IBM S/390, ya que la posición (registro en lugar de memoria) de cada operando debe especificarse separadamente en la instrucción. Volveremos sobre esta cuestión cuando veamos, en el siguiente capítulo, los formatos de instrucciones.

En términos de la acción del procesador, las operaciones de transferencia de datos son quizá las más sencillas. Cuando tanto el origen como el destino son registros, el procesador simplemente hace que los datos se transfieran de un registro a otro; esta es una operación interna al procesador. Si uno o ambos operandos están en memoria, el procesador debe realizar alguna o todas las tareas siguientes:

1. Calcular la dirección de memoria basándose en el modo de direccionamiento utilizado (Capítulo 11).
2. Si la dirección hace referencia a memoria virtual, traducir de dirección virtual a real.
3. Determinar si el elemento direccionado está en la caché.
4. Si no, cursar la orden al módulo de memoria.

ARITMÉTICAS

La mayoría de las máquinas proporcionan las operaciones aritméticas básicas de suma, resta, multiplicación y división. Estas se tienen siempre para números enteros con signo (en coma fija). A menudo las proporcionan también para números en coma flotante y para decimales empaquetados.

Entre otras operaciones posibles hay varias instrucciones de un solo operando; por ejemplo:

- ***Absolute*:** obtiene el valor absoluto del operando.
- ***Negate*:** cambia el signo del operando.
- ***Increment*:** incrementa en 1 el operando.
- ***Decrement*:** decrementa en 1 el operando.

La ejecución de una instrucción aritmética puede implicar operaciones de transferencia de datos para ubicar los operandos como entradas a la ALU, y para almacenar la salida de la ALU. La Figura 3.5 ilustra las transferencias involucradas tanto en operaciones de transferencia como en aritméticas. Por supuesto que, además, la parte ALU del procesador debe realizar la operación deseada.

LÓGICAS

La mayoría de las máquinas también proporcionan diversas operaciones para manipular bits individuales dentro de una palabra o de otra unidad direccionable. Están basadas en operaciones booleanas (*véase* Apéndice B).

La Tabla 10.6 muestra algunas de las operaciones lógicas básicas que pueden realizarse con datos booleanos o binarios. La operación NOT invierte un bit. Las funciones lógicas más comunes con dos operandos son la AND, la OR y la OR-exclusiva (XOR). La EQUAL (igual) es una comprobación binaria bastante útil.

Las operaciones lógicas pueden aplicarse desde bit a bit hasta unidades lógicas de datos de *n* bits. Así, si dos registros contienen los datos:

(R1) = 10100101

(R2) = 00001111

entonces:

(R1) AND (R2) = 00000101

donde la notación (X) significa el contenido de la posición X. Por lo tanto, la operación AND puede utilizarse como *máscara* para seleccionar ciertos bits de una palabra, poniendo a cero los restantes bits. En otro ejemplo, si dos registros contienen:

(R1) = 10100101

(R2) = 11111111

entonces:

(R1) XOR (R2) = 01011010

Con una palabra puesta a todo unos, la operación XOR invierte todos los bits de la otra palabra (complemento a uno).

Además de las operaciones lógicas bit a bit, la mayoría de las máquinas ofrecen diversas funciones de desplazamiento y rotación. Las operaciones más básicas se ilustran en la Figura 10.5. En un **desplazamiento lógico** se desplazan a la derecha o a la izquierda los bits de la palabra. En un extremo, el bit saliente al desplazar se pierde. En el otro extremo se introduce un 0. Los desplazamientos lógicos son útiles principalmente para aislar campos dentro de una palabra. Los ceros que se van introduciendo en la palabra desplazan la información no deseada que se va perdiendo por el otro extremo.

Tabla 10.6. Operaciones lógicas básicas.

P	Q	NOT P	PAND Q	P OR Q	P XOR Q	P = Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

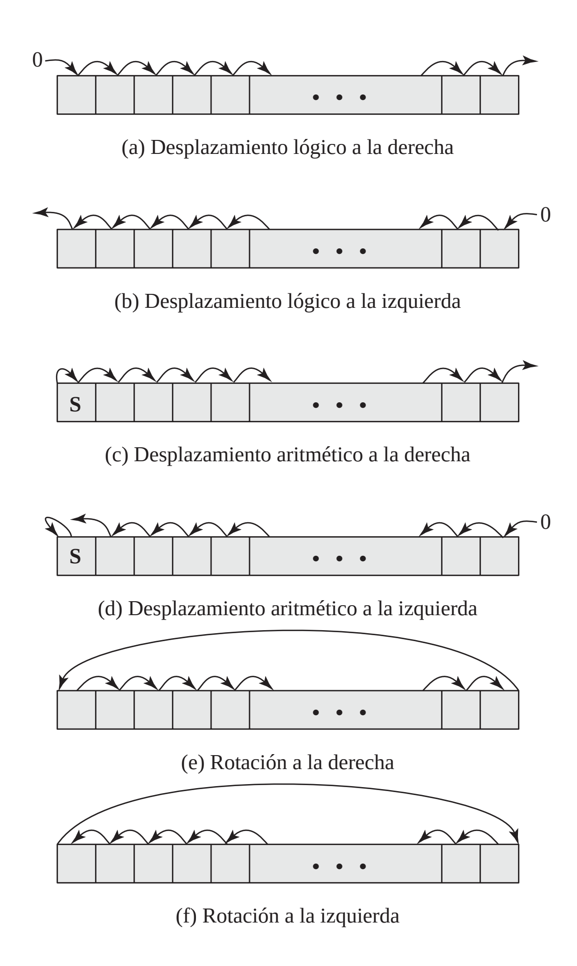


Figura 10.5. Operaciones de desplazamiento y de rotación.

Como ejemplo, suponga que queremos transmitir caracteres de datos, carácter a carácter, a un dispositivo de E/S. Si las palabras de memoria son de 16 bits y contienen dos caracteres, debemos *desempaquetar* los caracteres antes de enviarlos. Para enviar los dos caracteres de una palabra, procedemos como sigue:

1. Cargar la palabra en un registro.
2. Realizar la AND con el valor 1111111100000000. Esto enmascara el carácter de la derecha.
3. Desplazar el registro ocho veces. Esto desplaza el otro carácter a la mitad derecha del registro.
4. Realizar la E/S. El módulo de E/S leerá los 8 bits de orden más bajo del bus de datos.

Los pasos anteriores producen el envío del carácter de la izquierda. Para enviar el carácter de la derecha:

1. Cargar de nuevo la palabra en el registro.
2. AND con 0000000011111111.
3. Realizar la E/S.

La operación de **desplazamiento aritmético** trata el dato como entero con signo y no desplaza el bit de signo. En un desplazamiento aritmético a la derecha, el bit de signo normalmente se replica en la posición de bit de su derecha. En un desplazamiento aritmético a la izquierda, se realiza un desplazamiento lógico de todos los bits, exceptuando el de signo, que se mantiene. Estos desplazamientos pueden acelerar ciertas operaciones aritméticas. Con números en notación de complemento a dos, un desplazamiento aritmético a la derecha equivale a dividir entre dos, truncando los números impares. Un desplazamiento aritmético o uno lógico a la izquierda equivalen a multiplicar por dos, siempre y cuando no se produzca desbordamiento. Cuando hay desbordamiento, el resultado de la operación aritmética o la lógica son diferentes, pero el desplazamiento aritmético mantiene el signo del número. Debido a la posibilidad de desbordamiento, muchos procesadores, inccluidos el PowerPC o el Itanium, no incorporan esta operación. Otros, como el IBM S/390, sí la incluyen. Curiosamente, la arquitectura Pentium incorpora el desplazamiento aritmético a la izquierda, pero lo define de forma idéntica al desplazamiento lógico a la izquierda.

La **rotación**, o desplazamiento cíclico, preserva todos los bits con los que se está operando. Un posible uso de la rotación es ir volcando sucesivamente cada bit en la posición más a la izquierda, donde pueda ser identificado comprobando el bit de signo del dato (tratado este como número).

Como en el caso de operaciones aritméticas, las operaciones lógicas implican actividad de la ALU y pueden involucrar operaciones de transferencia de datos. La Tabla 10.7 incluye ejemplos de todas las operaciones de desplazamiento y de rotación discutidas en esta subsección.

CONVERSIÓN

Las instrucciones de conversión son aquellas que cambian el formato u operan sobre el formato de los datos. Un ejemplo es la conversión de decimal a binario. Un ejemplo de instrucción de edición más compleja es la instrucción de traducción, *Translate* (TR), del S/390. Esta instrucción puede utilizarse para convertir de un código de ocho bits a otro, y tiene tres operandos:

TR R1, R2, L

El operando R2 contiene la dirección de comienzo de una tabla de códigos de ocho bits. Se traducen los L bytes que comienzan en la dirección especificada en R1, sustituyéndose cada byte por el contenido del elemento de la tabla indexado por dicho byte. Por ejemplo, para traducir de EBCDIC a

Tabla 10.7. Ejemplo de operaciones de desplazamiento y de rotación.

Entrada	Operación	Resultado
10100110	Desplazamiento lógico a la derecha (3 bits)	00010100
10100110	Desplazamiento lógico a la izquierda (3 bits)	00110000
10100110	Desplazamiento aritmético a la derecha (3 bits)	11110111
10100110	Desplazamiento aritmético a la izquierda (3 bits)	10110000
10100110	Rotación a la derecha (3 bits)	11010100
10100110	Rotación a la izquierda (3 bits)	00110101

ASCII, primero creamos una tabla de 256 bytes en posiciones de memoria, por ejemplo, desde la 1000 a la 10FF. Esta tabla debe contener los caracteres del código ASCII en la secuencia de la representación binaria del código EBCDIC; es decir, el código de cada carácter ASCII se coloca en la tabla en una posición relativa igual al valor binario del código EBCDIC del mismo carácter. Así pues, las posiciones 10F0 a 10F9 contendrán los valores 30 a 39, ya que F0 es el código EBCDIC del dígito 0, cuyo código ASCII es el 30; y así hasta la posición correspondiente al dígito 9. Suponga ahora que tenemos la secuencia de dígitos 1984, en EBCDIC, a partir de la posición 2100, y queremos convertirla a ASCII. Suponga lo siguiente:

- Las posiciones 2100-2103 contienen F1 F9 F8 F4.
- R1 contiene 2100.
- R2 contiene 1000.

Entonces, si ejecutamos:

TR R1, R2, 4

Los contenidos de las posiciones 2100 a 2103 serían 31 39 38 34.

ENTRADA/SALIDA

Las instrucciones de entrada/salida se trataron con cierto detalle en el Capítulo 7. Como vimos, existen aproximaciones muy diversas, incluyendo entradas/salidas programadas aisladas, entradas/salidas programadas asignadas en memoria, DMA, y el uso de un procesador de E/S. Muchas implementaciones ofrecen solo unas pocas instrucciones de E/S, con acciones específicas indicadas mediante parámetros, códigos o palabras de órdenes.

CONTROL DEL SISTEMA

Las instrucciones de control del sistema son por lo general instrucciones privilegiadas que pueden ejecutarse solo mientras el procesador está en un estado privilegiado concreto, o está ejecutando un programa de una zona privilegiada específica de memoria. Normalmente, estas instrucciones están reservadas para que las use el sistema operativo.

Algunos ejemplos de operaciones de control del sistema son los siguientes. Una instrucción de control del sistema puede leer o alterar un registro de control; los registros de control se verán en el Capítulo 12. Otro ejemplo es una instrucción para leer o modificar una clave de protección de memoria, tal como la utilizada en el sistema de memoria del S/390. Otro ejemplo es acceder a bloques de control de procesos en un sistema con multiprogramación.

CONTROL DE FLUJO

En todos los tipos de operaciones discutidos hasta ahora, la siguiente instrucción a ejecutar es la inmediatamente posterior, en memoria, a la instrucción en curso. Sin embargo, una fracción

significativa de las instrucciones de cualquier programa tienen como misión cambiar la secuencia de ejecución normal. Para estas instrucciones, la operación que realiza el procesador es actualizar el contador de programa para que contenga la dirección de alguna de las instrucciones que hay en memoria.

Hay varias razones por las que se necesitan las operaciones de control de flujo o de transferencia del control. Entre las más importantes están:

1. En el uso práctico de los computadores es esencial poder ejecutar cada instrucción más de una vez, y puede que muchos miles de veces. Implementar una aplicación puede requerir miles o incluso millones de instrucciones. Esto sería impensable si hubiera que escribir cada instrucción por separado. Si se va a procesar una tabla o una lista de elementos, lo normal es utilizar un bucle de programa. Así una secuencia de instrucciones se ejecutaría repetidas veces para procesar todos los datos.

2. Prácticamente todos los programas implican algunas tomas de decisiones. Queremos que el computador haga algo cuando se cumple una condición, y otra cosa distinta si se cumple otra condición. Por ejemplo, una secuencia de instrucciones calcula la raíz cuadrada de un número. Al principio de la secuencia se comprueba el signo del número. Si este es negativo, el cálculo no se realiza, sino que se señala una condición de error.

3. Redactar correctamente un programa largo, o incluso uno de extensión moderada, es una tarea excesivamente compleja. Es de gran ayuda partir la tarea en trozos más pequeños con los que se trabaje por separado.

Retomamos la discusión sobre las operaciones de control de flujo que se pueden encontrar en repertorios de instrucciones: bifurcación, salto implícito y llamada a procedimiento.

Instrucciones de bifurcación. Una instrucción de bifurcación, también llamada de salto, tiene como uno de sus operandos la dirección de la siguiente instrucción a ejecutar. Las más frecuentes son las instrucciones de **salto condicional.** Es decir, se efectúa la bifurcación (se actualiza el contador de programa con la dirección especificada en el operando) solo si se cumple una condición dada. En caso contrario, se ejecuta la instrucción siguiente de la secuencia (se incrementa el contador de programa de la forma habitual). Una instrucción de salto en que se produce siempre la bifurcación es un **salto incondicional.**

Hay dos formas comunes de generación de la condición a comprobar en una instrucción de salto condicional. En primer lugar, la mayoría de las máquinas proporcionan un código de condición de uno o varios bits, que se actualiza cuando se ejecutan algunas operaciones. Este código puede imaginarse como un pequeño registro visible para el usuario. Como ejemplo, una operación aritmética (ADD, SUBTRACT, etc.) podría fijar un código de condición de dos bits, con uno de los cuatro valores siguientes: 0, positivo, negativo, desbordamiento. En tal caso, la máquina podría tener cuatro instrucciones de bifurcación o salto condicional:

BRP X Saltar a la posición X si el resultado es positivo.

BRN X Saltar a la posición X si el resultado es negativo.

BRZ X Saltar a la posición X si el resultado es cero.

BRO X Saltar a la posición X si se ha producido desbordamiento.

En todos los casos anteriores, el resultado al que se hace referencia es el de la última operación ejecutada que afecte al código de condición.

Otra aproximación que puede utilizarse con un formato de instrucción de tres direcciones consiste en realizar la comparación y especificar la bifurcación en la misma instrucción. Por ejemplo,

BRE R1,R2,X Saltar a X si el contenido de R1 es igual al de R2.

La Figura 10.6 muestra ejemplos de estas operaciones. Observe que un salto puede ser bien *hacia adelante* (a una instrucción con dirección más alta) o *hacia atrás* (dirección más baja). El ejemplo muestra cómo se pueden utilizar un salto incondicional y otro condicional para crear un bucle de repetición de instrucciones. Las instrucciones en las posiciones 202 a 210 se ejecutarán repetidas veces hasta que el resultado de restar Y de X sea 0.

Instrucciones de salto implícito. Otra forma común de instrucciones de control de flujo es la instrucción de salto implícito (*skip*). Esta instrucción incluye una dirección de manera implícita. Normalmente, el salto implícito implica que se va a saltar una instrucción; por lo tanto, la dirección implícita es igual a la dirección de la siguiente instrucción más la longitud de una instrucción.

Dado que la instrucción de salto implícito no requiere un campo de dirección de destino, este queda libre para otras cosas. Un ejemplo típico es la instrucción «incrementar y saltar si es cero» (ISZ). Considere el fragmento de programa siguiente:

```
301
•
•
•
309    ISZ    R1
310    BR     301
311
```

Las dos instrucciones de control de flujo se emplean para implementar un bucle iterativo. R1 se fija a un valor negativo, el opuesto del número de iteraciones a realizar. Al final del bucle, R1 se

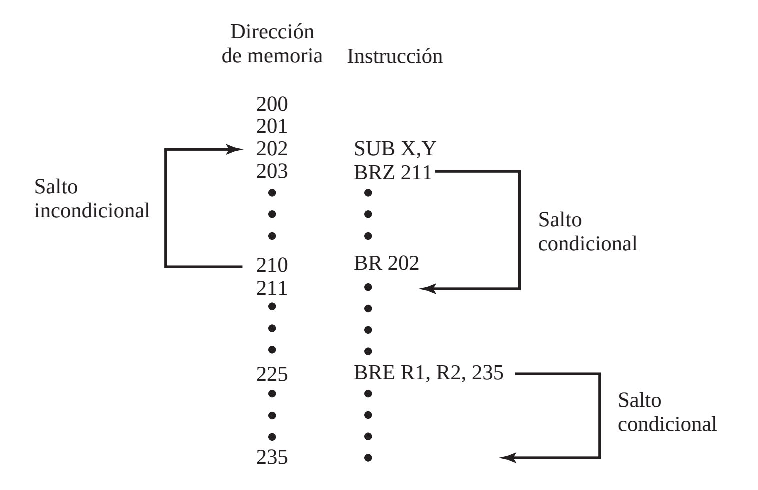


Figura 10.6. Instrucciones de bifurcación.

incrementa. Si es distinto de cero, el programa bifurca hacia el comienzo del bucle. Si no, se salta la instrucción de bifurcación y el programa continúa con la instrucción siguiente a la de fin del bucle.

Instrucciones de llamada a procedimiento. El *procedimiento (procedure)* fue quizá la innovación más importante en el desarrollo de los lenguajes de programación. Un procedimiento es un programa con entidad propia que se incorpora en un programa más grande. En cualquier punto del programa se puede invocar o *llamar* al procedimiento. Es decir, se ordena al procesador que pase a ejecutar el procedimiento completo y retornar después al punto en que tuvo lugar la llamada.

Las dos razones principales para el uso de los procedimientos son la economía y la modularidad. Un procedimiento permite que la misma porción de código se utilice muchas veces. Esto es importante para economizar en esfuerzo de programación y para hacer el uso más eficiente del espacio de memoria del sistema (el programa hay que almacenarlo). Los procedimientos permiten también que los programas largos se puedan subdividir en unidades más pequeñas. Este uso de la *modularidad* facilita enormemente la tarea de programar.

La utilización de procedimientos requiere de dos instrucciones básicas: una instrucción de llamada (*Call*), que produce una bifurcación desde la posición actual al procedimiento; y una instrucción de retorno del procedimiento (*Return*) al lugar desde el que se llamó. Ambas son modalidades de instrucciones de bifurcación.

La Figura 10.7a ilustra el uso de procedimientos para construir un programa. En este ejemplo, hay un programa principal que empieza en la posición 4000. Este programa incluye una llamada al procedimiento PROC1, que comienza en la posición 4500. Cuando se encuentra esta instrucción de llamada, el procesador interrumpe la ejecución del programa principal e inicia la ejecución de PROC1 captando la siguiente instrucción de la posición 4500. Dentro de PROC1, hay dos llamadas a PROC2, el cual comienza en la posición 4800. En cada caso, se interrumpe la ejecución de PROC1 y se ejecuta PROC2. La sentencia RETURN hace que el procesador vuelva al programa de llamada y continúe con la ejecución de la instrucción que sigue a la correspondiente CALL. Este comportamiento se ilustra en la Figura 10.7.b.

Merece la pena resaltar varios puntos:

1. Un procedimiento puede llamarse desde distintas posiciones.
2. Un procedimiento puede contener llamadas a otros procedimientos. Esto posibilita el *anidamiento* de procedimientos hasta una profundidad arbitraria.
3. Cada llamada a procedimiento está emparejada con un retorno en el programa llamado.

Ya que debe permitirse que el procedimiento se llame desde distintos puntos, el procesador debe preservar la dirección de retorno en algún sitio, para que este pueda realizarse correctamente. Hay tres lugares habituales para guardar la dirección de retorno:

- Un registro.
- Al principio del procedimiento.
- En la cabecera de la pila.

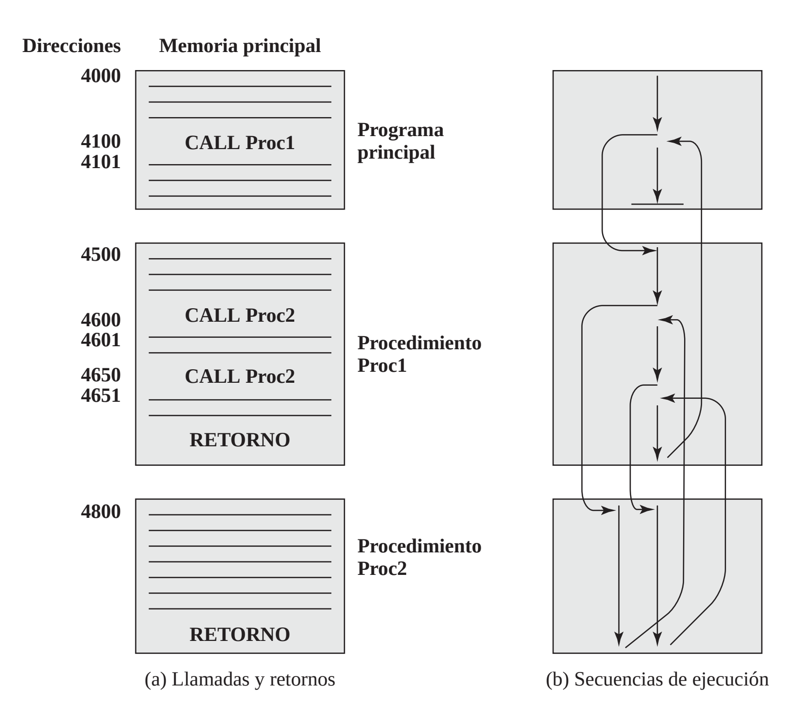


Figura 10.7. Procedimientos anidados.

Consideremos la instrucción en lenguaje máquina CALL X, que significa: *llamada al procediminento de la posición X*. Si se utiliza un registro, la ejecución de CALL X produce las siguientes acciones:

$$RN \longleftarrow PC + \Delta$$
$$PC \longleftarrow X$$

en donde RN es un registro que se utiliza siempre para este fin, PC es el contador de programa, y Δ es la longitud de la instrucción. El procedimiento llamado puede ahora preservar el contenido de RN para utilizarlo en el retorno posterior.

Una segunda posibilidad es almacenar la dirección de retorno al comienzo del procedimiento. En este caso, CALL X hace que:

$$X \longleftarrow PC + \Delta$$
$$PC \longleftarrow X + 1$$

La dirección de retorno queda almacenada en un lugar seguro.

Las dos aproximaciones anteriores son correctas y se han utilizado en la práctica. La única limitación que presentan es que impiden el uso de procedimientos *reentrantes*. Un procedimiento reentrante es aquel para el que es posible tener abiertas varias llamadas al mismo tiempo. Los procedimientos recursivos son un ejemplo del uso de esta característica.

Una aproximación más general y potente es utilizar una pila (*véase* el Apéndice 10A para una definición de la pila). Cuando el procesador ejecuta una llamada, coloca la dirección de retorno en la pila. Cuando ejecuta un retorno, utiliza la dirección almacenada en la pila. La Figura 10.8 ilustra la utilización de la pila.

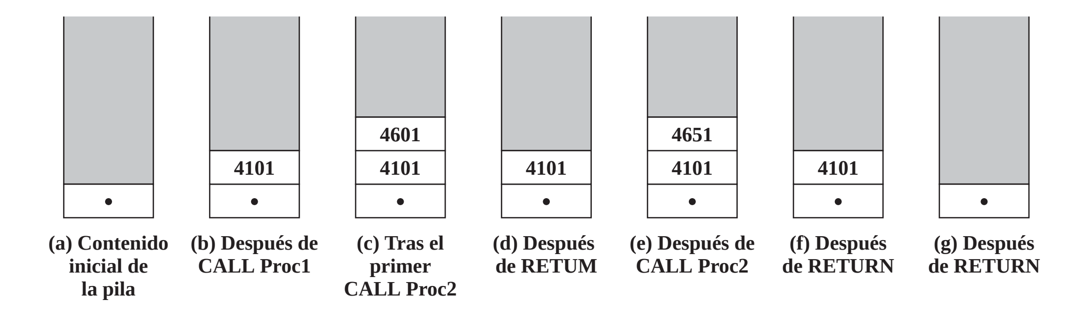


Figura 10.8. Uso de una pila para implementar el anidamiento de subrutinas de la Figura 10.7.

Además de aportar la dirección de retorno, es a menudo necesario pasar o transferir ciertos parámetros en la llamada a un procedimiento. Estos se pueden transferir mediante registros. Otra posibilidad es almacenar estos parámetros en memoria justo después de la instrucción CALL. En este caso, el retorno debe hacerse a la posición siguiente a los parámetros. De nuevo, estas dos aproximaciones presentan limitaciones. Si se emplean los registros, el programa llamado y el que hace la llamada deben escribirse de manera que se asegure el uso correcto de los registros. El almacenamiento de los parámetros en memoria hace difícil transferir un número variable de ellos. Ambas alternativas impiden el uso de procedimientos reentrantes.

Una alternativa más flexible para el paso de parámetros es la pila. Cuando el procesador ejecuta una llamada, no solo introduce en la pila la dirección de retorno, sino también los parámetros que deben transferirse al procedimiento llamado. El procedimiento invocado puede acceder a los parámetros en la pila. Para el retorno, los parámetros de retorno pueden introducirse también en la pila. El conjunto de parámetros completo que se almacena en la llamada a un procedimiento, incluyendo la dirección de retorno, se denomina *marco de pila*.

En la Figura 10.9 se muestra un ejemplo. En este ejemplo se hace referencia al procedimiento P, en el que se declaran las variables locales x1 y x2, y al procedimiento Q, que puede ser llamado por P, y en el que se declaran las variables locales y1 e y2. En la figura, el punto de retorno para cada procedimiento está en el primer elemento memorizado del correspondiente marco de pila. A continuación se almacena un puntero hacia el comienzo del marco precedente. Esto es necesario cuando el número o la longitud de los parámetros a introducir en la pila son variables.

10.5. TIPOS DE OPERACIONES EN EL PENTIUM Y EL POWERPC

TIPOS DE OPERACIONES DEL PENTIUM

El Pentium ofrece un amplio abanico de tipos de operaciones, incluyendo diversas instrucciones especializadas. Con ello se ha intentado dotar de medios a los programadores de compiladores para producir traducciones óptimas, a lenguaje máquina, de los programas en lenguaje de alto nivel. La Tabla 10.8 resume los distintos tipos y da ejemplos de cada uno. La mayoría coinciden con instrucciones convencionales que se pueden encontrar en la mayoría de los repertorios de instrucciones máquina, pero algunos de los tipos de instrucciones están adaptados a las arquitecturas 80x86/Pentium y son de particular interés.

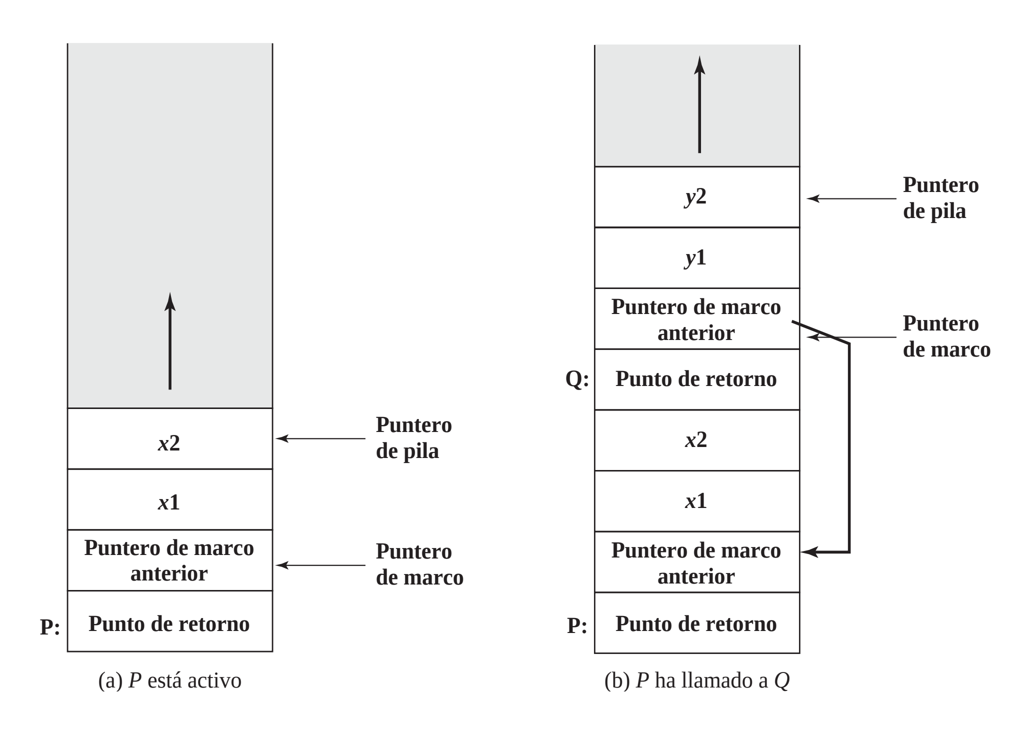


Figura 10.9. Crecimiento del marco de pila utilizando los procedimientos de ejemplo P y Q.

Tabla 10.8. Tipos de operaciones del Pentium (con ejemplos de operaciones típicas).

Instrucción	Descripción
Transferencias de datos	
MOV	Transferir el operando, entre registros o entre registro y memoria
PUSH	Apilar el operando
PUSHA	Apilar todos los registros
MOVSX	«Move byte,word,dword,sign extended». Transfiere un byte a una palabra, o una palabra a una palabra doble, extendiendo hacia la izquierda el signo según la representación en complemento a dos
LEA	«Load effective addres». Carga el desplazamiento del operando origen, en lugar de su valor, en el operando destino
XLAT	«Table lookup translation». Sustituye un byte de AL con otro de una tabla de traducción codificada por el usuario. Cuando se ejecuta XLAT, AL debe tener un índice sin signo de la tabla. XLAT cambia el índice que contiene AL por el correspondiente elemento de la tabla
IN, OUT	Entrada o salida de operando desde el (o al) espacio de E/S

(*continúa*)

Tabla 10.8. Tipos de operaciones del Pentium (con ejemplos de operaciones típicas) *(continuación).*

Instrucción	Descripción
Aritméticas	
ADD	Sumar los operandos
SUB	Restar los operandos
MUL	Multiplicación de enteros sin signo, con operandos de un byte, una palabra o una palabra doble, y como resultado una palabra, una palabra doble, o una cuádruple
IDIV	División con signo
Lógicas	
AND	Producto lógico de los operandos
BTS	«Bit Test and Set». Opera con un operando de campo de bits. La instrucción copia el valor actual de un bit en un indicador CF y pone a 1 el bit original
BSF	«Bit Scan Forward». Busca en una palabra o en una palabra doble el primer bit a 1 y almacena su número de posición en un registro
SHL/SHR	Desplazamiento lógico a izquierda o a derecha
SAL/SAR	Desplazamiento aritmético a izquierda o a derecha
ROL/ROR	Rotar a izquierda o a derecha
SETcc	Pone un byte a cero o a uno dependiendo de alguna de las 16 condiciones definidas por los indicadores de estado
Control de flujo	
JMP	Salto incondicional
CALL	Transfiere el control a otra posición. Antes de transferirlo, se introduce en la pila la dirección de la instrucción siguiente a la CALL
JE/JZ	Salto si igual/cero
LOOPE/LOOPZ	Bucle si igual/cero. Se trata de una bifurcación condicional utilizando un valor almacenado en el registro ECX. La instrucción primero decrementa ECX antes de comprobar ECX para la condición del salto
INT/INTO	Interrupción/Interrupción_si_desbordamiento. Transfiere el control a una rutina de servicio de interrupciones
Operaciones con cadenas	
MOVS	«Move byte, word, dword string». Esta instrucción opera con un elemento de una cadena, indexada por los registros ESI y EDI. Después de cada operación con un elemento de la cadena, dichos registros se incrementan o decrementan automáticamente, para apuntar al siguiente elemento de la cadena
LODS	Carga un byte, una palabra, o una palabra doble de una cadena

(*continúa*)

Tabla 10.8. Tipos de operaciones del Pentium (con ejemplos de operaciones típicas) *(continuación).*

Instrucción	Descripción
Instrucciones de soporte a lenguajes de alto nivel	
ENTER	Crea un marco de pila que puede utilizarse para implementar las reglas de un lenguaje de alto nivel estructurado por bloques
LEAVE	Realiza la función inversa de la instrucción ENTER
BOUND	Comprueba los extremos de una matriz. Verifica si el valor del operando 1 está entre ciertos límites inferior y superior. Los limites se encuentran en dos posiciones adyacentes indicadas por el operando 2. Si el valor está fuera de dichos límites se produce una interrupción. Esta instrucción se emplea para comprobar los índices de una matriz
Control de indicadores	
STC	Activa el indicador de acarreo
LAHF	Carga el registro A con los valores de los indicadores. Copia los bits SF, ZF, AF, PF, y CF en el registro A
Registro de segmentos	
LDS	Carga un puntero en el registro de segmentos D
Control del sistema	
HLT	Parar
LOCK	Toma posesión de la memoria compartida para que el Pentium haga uso exclusivo de ella durante la instrucción que sigue inmediatamente a la LOCK
ESC	Escape de ampliación del procesador. Es un código de escape para indicar que las instrucciones que siguen van a ser ejecutadas por un coprocesador aritmético que permite cálculos de alta precisión con enteros y en coma flotante
WAIT	Espera hasta un BUSY# negado. Suspende la ejecución del programa del Pentium hasta que el procesador detecte que el terminal BUSY está inactivo, indicando que el coprocesador aritmético ha finalizado su operación
Protección	
SGDT	Memoriza una tabla global de descriptores
LSL	Carga el límite de segmento. Carga un registro especificado por el usuario con un límite de segmento
VERR/VERW	Verifica el segmento para lectura/escritura.
Gestión de caché	
INVD	Limpia la memoria caché interna
WBINVD	Limpia la memoria caché interna después de escribir en memoria las líneas que han sido modificadas
INVLPG	Invalida una entrada al buffer TLB

INSTRUCCIONES DE LLAMADA/RETORNO

El Pentium contiene cuatro instrucciones para ejecutar llamadas/retornos a/de procedimientos: CALL, ENTER, LEAVE, RETURN. Es instructivo analizar las posibilidades que ofrecen estas instrucciones. Recuerde, de la Figura 10.9, que una forma habitual de implementar el esquema de llamada/retorno es mediante el uso de marcos de pila. Cuando se invoca un nuevo procedimiento, al entrar en el mismo, debe realizarse lo siguiente:

- Introducir en la pila (apilar) el punto de retorno.
- Apilar también el puntero de marco actual.
- Copiar el puntero de pila como nuevo valor para el puntero de marco.
- Ajustar el puntero de pila para asignar un marco.

La instrucción CALL apila el valor actual del puntero de instrucciones y provoca un salto al punto de entrada del procedimiento, colocando la dirección del punto de entrada en el puntero de instrucciones. En el 8088 y el 8086, el procedimiento solía iniciarse con la secuencia:

```
PUSH    EBP
MOV     EBP, ESP
SUB     ESP, espacio_para_variables_locales
```

en donde EBP es el puntero de marco y ESP es el puntero de pila. En el 80286 y máquinas posteriores, todas las operaciones anteriores se realizan con una única instrucción, la ENTER.

La instrucción ENTER se añadió al repertorio para dar facilidad al compilador. Esta instrucción incluye también una forma de implementar los denominados procedimientos anidados de lenguajes tales como Pascal, COBOL y Ada (no se dan en el C o el FORTRAN). Existen formas mejores de manipular llamadas a procedimientos anidados para estos lenguajes. Además, aunque la instrucción ENTER ahorra unos cuantos bytes de memoria en comparación con la secuencia PUSH, MOV, SUB (cuatro bytes en lugar de seis), tarda realmente más en ejecutarse (diez ciclos de reloj en lugar de seis ciclos). En consecuencia, aunque haya parecido una buena idea de los diseñadores del repertorio de instrucciones introducir esta instrucción, complica la implementación del procesador y proporciona poco o ningún beneficio. Ya veremos que, por contra, una alternativa de diseño RISC del procesador, evitaría instrucciones complejas como la ENTER, y podría producir una implementación más eficiente con una secuencia de instrucciones más sencillas.

Gestión de memoria. Otro conjunto de instrucciones especiales está dedicado a la segmentación de memoria. Son instrucciones privilegiadas que pueden ejecutarse solo desde el sistema operativo. Permiten cargar y leer tablas de segmentos locales y globales (llamadas tablas de descriptores), y comprobar y alterar el nivel de privilegio de un segmento.

Las instrucciones especiales para manejar la caché *on-chip* se discutieron en el Capítulo 4.

Códigos de condición. Hemos mencionado que los códigos de condición son bits de registros especiales que pueden ser activados por ciertas operaciones, y ser utilizados en instrucciones de

bifurcación condicional. Estos bits de condición se activan por operaciones aritméticas y de comparación. En la mayoría de los repertorios, la instrucción de comparación resta dos operandos, como hace la operación de resta, pero solo fija los bits de condición, mientras que la operación de resta almacena además el resultado de la resta en el operando de destino.

La Tabla 10.9 describe los códigos de condición utilizados en el Pentium. Cada condición, o una combinación de ellas, puede ser comprobada al objeto de efectuar un salto condicional. La Tabla 10.10 muestra las combinaciones de condiciones para las que se han definido codops de saltos condicionales.

Sobre ellas pueden hacerse varias observaciones interesantes. En primer lugar, podría interesarnos comprobar dos operandos numéricos para determinar si uno es mayor que otro. Pero esto va a depender de si los números tienen o no signo. Por ejemplo, el número de ocho bits 11111111 es mayor que el 00000000 si los dos números se interpretan como enteros sin signo ($255 > 0$), pero es menor si se consideran como números de ocho bits en complemento a dos ($-1 < 0$). Muchos lenguajes ensambladores introducen dos denominaciones diferentes que permiten distinguir entre los dos casos anteriores: si se están comparando dos números como enteros con signo, se emplean los términos *menor que* y *mayor que*; si se comparan como enteros sin signo, se emplean los términos *inferior* y *superior*.

Una segunda observación afecta a la complejidad de comparar enteros con signo. Un resultado con signo es mayor o igual que cero si (1) el bit de signo es cero y no ha habido desbordamiento ($S = 0$ AND $O = 0$), o (2) si el bit de signo vale uno y ha habido desbordamiento. Un análisis de la Figura 9.4 le convencerá de que son correctas las condiciones que se comprueban para las distintas operaciones (véase el Problema 10.13).

Instrucciones MMX del Pentium. Intel introdujo en 1996 la tecnología MMX en su línea de procesadores Pentium. MMX comprende instrucciones muy optimizadas para tareas multimedia.

Tabla 10.9. Códigos de condición del Pentium.

Bit de estado	Nombre	Descripción
C	Acarreo	Indica un acarreo o un acarreo negativo en la posición de bit más significativa, tras una operación aritmética. También se modifica por algunas operaciones de desplazamiento y de rotación.
P	Paridad	Paridad del resultado de una operación aritmética o lógica. Un 1 indica paridad par, y el 0 paridad impar.
A	Acarreo auxiliar	Representa un acarreo o un acarreo negativo entre las dos mitades de una operación aritmética o lógica de ocho bits, utilizando el registro AL.
Z	Cero	Indica que el resultado de una operación aritmética o lógica es 0.
S	Signo	Indica el signo del resultado de una operación aritmética o lógica.
O	Desbordamiento	Indica un desbordamiento aritmético después de una suma o una resta.

Tabla 10.10. Condiciones de bifurcación e instrucciones SETcc del Pentium.

Símbolo	Condición comprobada	Comentario
A, NBE	C = 0 AND Z = 0	Superior; no inferior o igual (en números con signo: mayor que)
AE, NB, NC	C = 0	Superior o igual; no inferior (mayor que o igual); no hay acarreo
B, NAE, C	C = 1	Inferior; no superior o igual (menor que); hay acarreo
BE, NA	C = 1 OR Z = 1	Inferior o igual; no superior (menor que o igual)
E, Z	Z = 1	Igual; cero (para números con o sin signo)
G, NLE	[(S = 1 AND O = 1) OR (S = 0 AND O = 0)] AND [Z = 0]	Mayor que; no menor que o igual (con signo)
GE, NL	(S = 1 AND O = 1) OR (S = 0 AND O = 0)	Mayor que o igual; no menor que (con signo)
L, NGE	(S = 1 AND O = 0) OR (S = 0 AND O = 1)	Menor que; no mayor que o igual (con signo)
LE, NG	(S = 1 AND O = 0) OR (S = 0 AND O = 1) OR (Z = 1)	Menor que o igual; no mayor que (con signo)
NE, NZ	Z = 0	Distinto; distinto de cero (con o sin signo)
NO	O = 0	No desbordamiento
NS	S = 0	No signo (no negativo)
NP, PO	P = 0	No paridad; paridad impar
O	O = 1	Desbordamiento
P	P = 1	Paridad; paridad par
S	S = 1	Signo (negativo)

Hay 57 instrucciones nuevas que tratan los datos en un modo SIMD (*single-instruction, multiple-data*), es decir una secuencia de instrucciones y múltiples secuencias de datos, que posibilita efectuar la misma operación, tal como una suma o una multiplicación, con varios elementos de datos a la vez. Cada instrucción suele ejecutarse en un ciclo de reloj. Para ciertas aplicaciones, estas operaciones rápidas en paralelo pueden conducir a una velocidad entre dos y ocho veces superior a la de algoritmos equiparables que no utilicen las instrucciones MMX [ATKI96].

El conjunto de instrucciones MMX está orientado a programación multimedia. Los datos de vídeo y audio suelen representarse mediante vectores o matrices grandes compuestos por datos de longitud reducida (8 o 16 bits), mientras que las instrucciones convencionales operan normalmente con datos de 32 o 64 bits. Estos son algunos ejemplos: en gráficos y en vídeo cada escena consiste en

una matriz de puntos de imagen [2], y hay ocho bits para cada punto de imagen u ocho bits para cada componente de color (rojo, verde, azul) del punto de imagen. Las muestras de audio suelen estar discretizadas con 16 bits. Para algunos algoritmos de gráficos 3D es común emplear 32 bits para los tipos de datos básicos. Para posibilitar el procesamiento paralelo con estos tamaños de datos, en MMX se definen tres nuevos tipos de datos. Tienen una longitud de 64 bits y constan de varios campos de datos más pequeños, cada uno de los cuales contiene un entero en coma fija. Estos tipos son los siguientes:

- **Byte empaquetado:** ocho bytes en una cantidad de 64 bits.
- **Palabra empaquetada:** cuatro palabras de 16 bits empaquetadas en 64 bits.
- **Palabra doble empaquetada:** dos palabras dobles de 32 bits empaquetadas en 64 bits.

La Tabla 10.11 resume el repertorio de instrucciones MMX. La mayoría de las instrucciones implica un procesamiento paralelo de bytes, palabras, o palabras dobles. Por ejemplo, la instrucción PSLLW efectúa un desplazamiento lógico hacia la izquierda de cada una de las cuatro palabras del operando (una palabra empaquetada); la instrucción PADDB toma como operando un byte empaquetado y realizada en paralelo sumas con cada posición de byte para producir un byte empaquetado de salida.

Una característica inusual que presenta el nuevo conjunto de instrucciones es la introducción de la aritmética con saturación. Con la aritmética sin signo ordinaria, cuando una operación produce un desbordamiento (es decir, se produce un acarreo en la posición de bit más significativa), el bit extra se trunca. Este tipo de truncamiento se conoce con el término inglés *wraparound*, y puede por ejemplo hacer que el resultado de una suma sea menor que los dos operandos de entrada. Considere por ejemplo las dos palabras en hexadecimal F000h y 3000h. Su suma se expresaría como:

$$\begin{array}{rr} \text{F000h} = & 1111\ 0000\ 0000\ 0000 \\ +\text{3000h} = & \underline{0011\ 0000\ 0000\ 0000} \\ & 10010\ 0000\ 0000\ 0000 = \text{2000h} \end{array}$$

Si los dos números representaban intensidad de imagen, el resultado de la suma hace que la combinación de zonas sombreadas oscuras aparezca como más clara. Esto no es lo que se pretende normalmente. Mediante la aritmética con saturación, cuando la suma produce un desbordamiento, o la resta produce un desbordamiento negativo, el resultado se fija respectivamente al mayor o al menor valor representable. Para el ejemplo anterior, la aritmética con saturación daría como resultado:

$$\begin{array}{rr} \text{F000h} = & 1111\ 0000\ 0000\ 0000 \\ +\text{3000h} = & \underline{0011\ 0000\ 0000\ 0000} \\ & 10010\ 0000\ 0000\ 0000 \\ & 1111\ 1111\ 1111\ 1111 = \text{FFFh} \end{array}$$

Para apreciar el uso de las instrucciones MMX, vamos a considerar un ejemplo tomado de [PELE97]. Una función típica en vídeo es el efecto de desvanecimiento o extinción progresiva (*fade-out*) y reaparición (*fade-in*), mediante la cual una imagen se deshace y convierte gradualmente en otra. Las dos imágenes se combinan mediante una media ponderada:

$$\text{Pixel_resultado} = \text{Pixel_A} \times \text{fade} + \text{Pixel_B} \times (1 - \text{fade})$$

[2] Un píxel o punto de imagen es el elemento más pequeño de una imagen digital al que se puede asignar un nivel de gris. En otras palabras, un píxel es cada punto individual de una imagen representada mediante una matriz de puntos.

Tabla 10.11. Repertorio de instrucciones MMX.

Categoría	Instrucción	Descripción
Aritméticas	PADD [B, W, D]	Suma paralela de ocho bytes empaquetados, cuatro palabras de 16 bits, o dos palabras dobles de 32 bits, con truncamiento
	PADDS [B, W]	Suma con saturación
	PADDUS [B, W]	Suma sin signo con saturación
	PSUB [B, W, D]	Resta con truncamiento
	PSUBS [B, W]	Resta con saturación
	PSUDUS [B, W]	Resta sin signo con saturación
	PMULHW	Multiplicación paralela de cuatro palabras con signo de 16 bits, con elección de los 16 bits más significativos del resultado de 32 bits.
	PMULLW	Multiplicación paralela de cuatro palabras con signo de 16 bits, con elección de los 16 bits menos significativos del resultado de 32 bits
	PMADDWD	Multiplicación paralela de cuatro palabras con signo de 16 bits; y suma a la vez pares adyacentes de los resultados de 32 bits
Comparación	PCMPEQ [B, W, D]	Comparación paralela de igualdad; el resultado es una máscara de unos cuando es verdadero, o una máscara de ceros si es falso
	PCMPGT [B, W, D]	Comparación paralela de magnitud: mayor que; el resultado es una máscara de unos cuando es verdadero, o una máscara de ceros si es falso
Conversión	PACKUSWB	Empaqueta palabras en bytes con saturación sin signo
	PACKSS [WB, DW]	Empaqueta palabras en bytes, o palabras dobles en palabras, con saturación con signo
	PUNPCKH [BW, WD, DQ]	Desempaqueta en paralelo (mezcla entrelazada) los bytes, palabras, o dobles palabras más significativas del registro MMX
	PUNPCKL [BW, WD, DQ]	Desempaqueta en paralelo (mezcla entrelazada) los bytes, palabras, o dobles palabras menos significativas del registro MMX
Lógicas	PAND	Producto lógico (AND bit a bit) de 64 bits
	PNDN	Producto lógico y complemento (NAND bit a bit) de 64 bits
	POR	Suma lógica (OR bit a bit) de 64 bits
	PXOR	OR exclusiva (XOR bit a bit) de 64 bits

(continúa)

Tabla 10.11. Repertorio de instrucciones MMX *(continuación).*

Categoría	Instrucción	Descripción
Desplazamiento	PSLL [W, D, Q]	Desplazamiento lógico a la izquierda en paralelo de palabras, palabras dobles o palabra cuádruple, tantas veces como se especifique en el registro MMX o mediante un valor inmediato
	PSRL [W, D, Q]	Desplazamiento lógico a la derecha en paralelo de palabras, palabras dobles, o palabra cuádruple
	PSRA [W, D]	Desplazamiento aritmético a la derecha en paralelo de palabras, palabras dobles, o palabra cuádruple
Transferencia de datos	MOV [D, Q]	Transfiere una palabra doble o una cuádruple a/desde el registro MMX
Gestión de estado	EMMS	Descarga el estado MMX (descarga los bits de etiqueta de los registros FP)

Nota: para aquellas instrucciones que soportan varios tipos de datos, estos se indican entre corchetes: [byte (B), palabra (W), palabra doble (D), palabra cuádruple (Q)].

Este cálculo se efectúa para cada posición de punto de imagen en A y B. Si se produce una secuencia de vídeo mientras *fade* está cambiando progresivamente desde 1 a 0 (con una escala ajustada a un entero de ocho bits), el resultado es una transformación paulatina de la imagen A en la B.

La Figura 10.10 muestra, para un conjunto de puntos de imagen, la secuencia de pasos necesaria. Las componentes de píxel de ocho bits son transformadas en elementos de 16 bits para adaptarlas al tamaño de las multiplicaciones MMX de 16 bits. Si estas imágenes tienen una resolución de 640×480, y en el cambio de imagen se emplean los 255 valores posibles de *fade*, se ejecutarían 535 millones de instrucciones cuando se emplea MMX. El mismo procesamiento realizado sin las instrucciones MMX requeriría 1,4 miles de millones de instrucciones [INTE98b].

TIPOS DE OPERACIONES DEL POWERPC

El PowerPC ofrece una gran variedad de tipos de operaciones. La Tabla 10.12 los enumera y proporciona algunos ejemplos. Merece la pena mencionar varias características.

Instrucciones dedicadas a bifurcaciones. En el PowerPC se contemplan las posibilidades usuales de bifurcación condicional e incondicional. Las instrucciones de bifurcación condicional comprueban un solo bit del registro de condición para determinar un estado de verdadero, falso o indiferencia, y el contenido de un registro de cuenta para comprobar condiciones de cero, distinto de cero o indiferencia. Así pues, en instrucciones de bifurcación condicional se admiten nueve posibles condiciones. Cuando se comprueba el estado de cero o distinto de cero del registro de cuenta, antes se decrementa en 1 su valor. Esto es conveniente al objeto de implementar bucles iterativos.

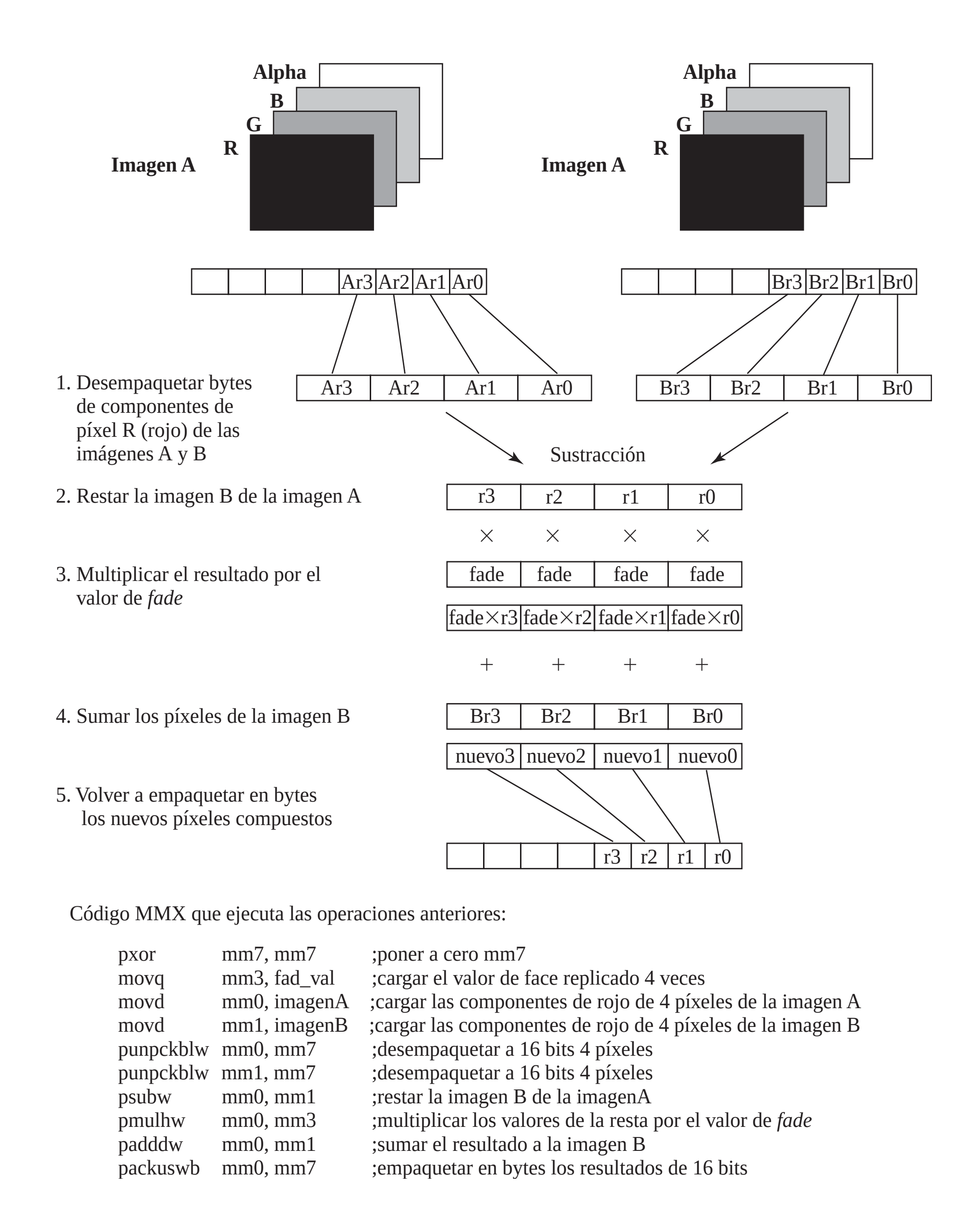


Figura 10.10. Composición de imagen en una representación de planos de colores [PELE97].

Las instrucciones de bifurcación pueden también indicar que la dirección de la posición siguiente a la bifurcación se guarde en el registro de enlace, descrito en el Capítulo 14. Esto facilita el procesamiento de llamadas/retornos.

Instrucciones de carga/memorización. En la arquitectura PowerPC, a memoria solo acceden las instrucciones de carga y memorización; las instrucciones aritméticas y lógicas se realizan solo con contenidos de registros. Esto es característico de un diseño RISC, y se verá con mayor detalle en el Capítulo 13.

Tabla 10.12. Tipos de operaciones del PowerPC (con ejemplos de operaciones típicas).

Instrucción	Descripción
Dedicadas a bifurcaciones	
b	Bifurcación o salto incondicional
bl	Saltar a la dirección destino y colocar en el registro de enlace la dirección efectiva de la instrucción siguiente a la bifurcación
bc	Salto condicional en función del registro de cuenta y/o bit del registro de condición
sc	Llamada al sistema para invocar un servicio del sistema operativo
trap	Comparar dos operandos y, si se cumple la condición especificada, invocar al gestor de interrupciones del sistema
Carga/Memorización	
lwzu	Cargar una palabra y poner a cero los restantes bits de la izquierda; actualizar el registro origen
ld	Cargar una palabra doble
lmw	Cargar una palabra múltiple; cargar palabras consecutivas en registros contiguos, desde el especificado hasta el registro 31 de uso general
lswx	Cargar una cadena de bytes en registros comenzando por el registro indicado; cuatro bytes por registro; y con conexión cíclica del registro 31 al 0
Aritmética de enteros	
add	Sumar los contenidos de dos registros y ubicar el resultado en un tercero
subf	Restar los contenidos de dos registros y ubicar el resultado en un tercero
mullw	Multiplicar los 32 bits menos significativos de los contenidos de dos registros y guardar el producto de 64 bits en un tercer registro
divd	Dividir los contenidos de 64 bits de dos registros y guardar el cociente en un tercer registro
Lógicas y Desplazamientos	
cmp	Comparar dos operandos y fijar cuatro bits de condición en el campo del registro de condición que se especifica
crand	AND del registro de condición: se calcula el producto lógico de dos bits del registro de condición y el resultado se guarda en una de las dos posiciones de bit
and	Producto lógico de los contenidos de dos registros y guardar el resultado en un tercero
cntlzd	Cuenta el número de bits a 0 consecutivos del registro origen, empezando por el bit cero, y guarda el resultado de la cuenta en el registro destino

(continúa)

Tabla 10.12. Tipos de operaciones del PowerPC (con ejemplos de operaciones típicas) *(continuación).*

Instrucción	Descripción
rldic	Rotar a la izquierda un registro de palabra doble, realizar la AND con una máscara, y almacenar el registro destino
sld	Desplazar a la izquierda el registro origen y almacenar en el registro de destino
Coma flotante	
lfs	Cargar de memoria un número en coma flotante de 32 bits, convertir al formato de 64 bits, y memorizar en un registro de coma flotante
fadd	Sumar los contenidos de dos registros y ubicar el resultado en un tercero
fmadd	Multiplicar los contenidos de dos registros, sumar el contenido de un tercero, y colocar el resultado en un cuarto registro
fcmpu	Comparar dos operandos en coma flotante y fijar los bits de condición
Gestión de caché	
dcbf	Limpia un bloque de la caché de datos; busca en las direcciones concretas de la caché y realiza la operación
icbi	Invalida un bloque de la caché de instrucciones

Hay dos características que distinguen a las diferentes instrucciones de carga/memorización:

- **Tamaño del dato:** los datos pueden transferirse en unidades de un byte, media palabra, palabra o palabra doble. También se dispone de instrucciones para cargar o memorizar una cadena de bytes en, o de, un conjunto registros.
- **Extensión del signo:** cuando se carga una palabra o una media palabra, los bits no utilizados de la parte izquierda del registro de 64 bits de destino se rellenan o bien con ceros o con el valor del bit de signo del dato cargado.

10.6. LENGUAJE ENSAMBLADOR

Un procesador puede interpretar y ejecutar instrucciones máquina. Estas instrucciones son simplemente números binarios almacenados en el computador. Si un programador quisiera programar directamente en lenguaje máquina, necesitaría introducir los programas como datos binarios.

Considere la sencilla sentencia BASIC:

$$N = I + J + K$$

Suponga que queremos programar esta sentencia en lenguaje máquina y dar a I, J, y K los valores iniciales 2, 3, y 4, respectivamente. La forma de hacer esto se muestra en la Figura 10.11a. El

Direción	Contenido			
101	0010	0010	101	2201
102	0001	0010	102	1202
103	0001	0010	103	1203
104	0011	0010	104	3204
201	0000	0000	201	0002
202	0000	0000	202	0003
203	0000	0000	203	0004
204	0000	0000	204	0000

(a) Programa en binario

Dirección	Contenido
101	2201
102	1202
103	1203
104	3204
201	0002
202	0003
203	0004
204	0000

(b) Programa en hexadecimal

Dirección	Instrucción	
101	LDA	201
102	ADD	202
103	ADD	203
104	STA	204
201	DAT	2
202	DAT	3
203	DAT	4
204	DAT	0

(c) Programa simbólico

Etiqueta	Operación	Operando
FORMUL	LDA	I
	ADD	J
	ADD	K
	STA	N
I	DATA	2
J	DATA	3
K	DATA	4
N	DATA	0

(d) Programa en ensamblador

Figura 10.11. Cálculo de la fórmula N = I + J + K.

programa empieza en la posición 101 (hexadecimal). Se reserva memoria para las cuatro variables a partir de la posición 201. El programa consta de cuatro instrucciones:

1. Cargar el contenido de la posición 201 en el acumulador (AC).
2. Sumar a AC el contenido de la posición 202.
3. Sumar a AC el contenido de la posición 203.
4. Memorizar el contenido de AC en la posición 204.

Esto es claramente un proceso tedioso y muy susceptible a errores.

Una ligera mejora consiste en redactar el programa en hexadecimal en lugar de en binario (Figura 10.11b). El programa se escribiría mediante una serie de líneas, en la que cada línea contiene la dirección de una posición de memoria y el código hexadecimal del valor binario a memorizar en tal posición. Necesitamos entonces un programa que acepte esta entrada, traduzca cada línea a un número binario y lo almacene en la posición especificada.

Para que la mejora sea más significativa, podemos hacer uso de nombres simbólicos o nemotécnicos de las instrucciones. El resultado es el *programa simbólico* mostrado en la Figura 10.11c. Cada línea sigue representando una posición de memoria y consta de tres campos separados por espacios. El primer campo contiene la dirección de una posición. En cada instrucción, el segundo campo contiene el símbolo de tres letras que representa su código de operación. Si se trata de una instrucción que hace referencia a memoria, un tercer campo contiene la dirección. Para memorizar un dato concreto en una posición dada, nos inventamos una *pseudoinstrucción* con el símbolo DAT. Esta es meramente un indicador de que el tercer campo de la línea contiene un número hexadecimal a memorizar en la posición que especifica el primer campo.

Para este tipo de descripción de entrada necesitamos un programa ligeramente más complejo. El programa aceptaría como entrada cada línea, generaría un número binario basándose en los campos segundo y tercero (si lo hay), y lo memorizaría en la posición indicada por el primer campo.

El uso de programas simbólicos hace la vida mucho más fácil pero es aún engorroso. En particular, hay que dar una dirección absoluta para cada palabra. Esto significa que el programa y los datos solo pueden cargarse en posiciones concretas de memoria, y que debemos saber siempre cuáles son. Peor aún, supongamos que se quiere algún día cambiar el programa para añadir o borrar una línea. Esto cambiaría las direcciones de todas las palabras siguientes.

Un procedimiento mejor, utilizado con frecuencia, es emplear direcciones simbólicas. Esto se ilustra en la Figura 10.11d. Cada línea sigue teniendo tres campos. El primero sigue siendo para la dirección, pero se utiliza un símbolo en lugar de una dirección numérica absoluta. Algunas líneas carecen de dirección, indicando que la dirección de dicha línea es uno más que la dirección de la precedente. Para las instrucciones que hacen referencia a memoria, el tercer campo contiene también una dirección simbólica.

Con este último refinamiento, hemos inventado un *lenguaje ensamblador*. Los programas escritos en lenguaje ensamblador (programas en ensamblador) se traducen a lenguaje máquina mediante un *ensamblador*. Este programa debe no solo realizar la traducción simbólica mencionada antes, sino también asignar direcciones de memoria a las direcciones simbólicas.

El desarrollo de los lenguajes ensambladores fue un logro importante en la evolución de la tecnología de los computadores. Fue el primer paso hacia los lenguajes de alto nivel utilizados hoy en día. Aunque son pocos los programadores que lo utilicen, prácticamente todos los procesadores disponen de lenguajes ensambladores. Se utilizan para programas del sistema tales como compiladores y rutinas de E/S.

10.7. LECTURAS RECOMENDADAS

El repertorio de instrucciones del Pentium está bien descrito en [BREY97]; y el del PowerPC en [IBM94] y [WEIS94].

BREY03 Brey, B.: *The Intel Microprocessors: 8086/8066, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4.* Upper Saddle River, NJ. Prentice Hall, 2000.

IBM94 International Business Machines, Inc.: *The PowerPC Architecture. A Specification for a New Family of RISC Processors.* San Francisco, CA. Morgan Kaufmann, 1994.

WEIS94 Weiss, S. y Smith, J.: *POWER and PowerPC.* San Francisco. Morgan Kaufmann, 1994.

10.8. PALABRAS CLAVE, PREGUNTAS DE REPASO Y PROBLEMAS

PALABRAS CLAVE

acumulador	*little endian* (del extremo menor)	retorno de procedimiento
dirección	desplazamiento lógico	*push* (introducir en la pila, «apilar»)
desplazamiento aritmético	instrucción máquina	procedimiento reentrante
bi-endian (de ambos extremos)	operando	notación polaca inversa
big endian (del extremo mayor)	operación	rotar
bifurcación	decimal empaquetado	salto implícito
bifurcación condicional	pop (extraer de la pila, «desapilar»)	pila
repertorio de instrucciones	llamada a procedimiento	
salto		

PREGUNTAS DE REPASO

10.1. ¿Cuáles son los componentes típicos de una instrucción máquina?

10.2. ¿Qué posiciones de memorización pueden contener operandos de origen y de destino?

10.3. Si una instrucción contiene cuatro direcciones ¿qué propósito podría tener cada dirección?

10.4. Enumere y explique brevemente cinco aspectos importantes en el diseño del repertorio de instrucciones.

10.5. ¿Qué tipos de operandos son usuales en los repertorios de instrucciones máquina?

10.6. ¿Qué relación existe entre el código de caracteres IRA o ASCII y la representación decimal empaquetada?

10.7. ¿Qué diferencia existe entre desplazamiento aritmético y desplazamiento lógico?

10.8. ¿Por qué son necesarias las instrucciones de control de flujo de ejecución?

10.9. Enumere y explique brevemente dos formas comunes de generar la condición a comprobar en una instrucción de bifurcación o salto condicional.

10.10. ¿Qué se entiende por *anidamiento de procedimientos*?

10.11. Enumere tres posibles ubicaciones para almacenar la dirección de retorno de un procedimiento.

10.12. ¿Qué es un procedimiento reentrante?

10.13. ¿Qué diferencia hay entre lenguaje ensamblador y lenguaje máquina?

10.14. ¿En qué consiste la notación polaca inversa?

10.15. ¿Cuál es la diferencia entre *big endian* y *little endian*?

PROBLEMAS

10.1. Exprese en notación hexadecimal:
a) El número en formato decimal empaquetado: 23.
b) La pareja de caracteres ASCII: 23.

10.2. Para cada uno de los siguientes números en decimal empaquetado, indique su valor decimal:
a) 0111 0011 0000 1001 *b)* 0101 1000 0010 *c)* 0100 1010 0110

10.3. Un determinado microprocesador tiene palabras de un byte. ¿Cuáles son los valores enteros más pequeño y más grande que pueden representarse en las siguientes representaciones?

a) Sin signo.
b) En signo-magnitud.
c) En complemento a uno.
d) En complemento a dos.
e) Decimal empaquetado sin signo.
f) Decimal empaquetado con signo.

10.4. Muchos procesadores incluyen lógica para realizar operaciones aritméticas con números en decimal empaquetado. Aunque las reglas para la aritmética decimal son similares a las de las operaciones binarias, los resultados decimales pueden requerir algunas correcciones o ajustes de los dígitos individuales cuando se emplea lógica binaria.

Considere la suma en decimal de dos números sin signo. Si cada número consta de N dígitos, habrá $4N$ bits en cada número. Los dos números se suman mediante un sumador binario. Sugiera una regla sencilla para corregir el resultado. Realice después la suma de los números 1698 y 1786.

10.5. El complemento a diez de un número decimal X se define como $10^N - X$, donde N es el número de dígitos decimales del número. Describa el uso de la representación en complemento a diez para realizar la resta decimal. Ilustre el procedimiento restando $(0326)_{10}$ de $(0736)_{10}$.

10.6. Compare las máquinas de cero, una, dos y tres direcciones, escribiendo programas que calculen:

$$X = (A + B (C) / (D - E (F)$$

para cada una de las cuatro máquinas. Las instrucciones de que se dispone son:

0 Direcciones	1 Dirección	2 Direcciones	3 Direcciones
PUSH M	LOAD M	MOVE (Z ← Y)	MOVE (X ← Y)
POP M	STORE M	ADD (X ← X + Y)	ADD (X ← Y + Z)
ADD	ADD M	SUB (X ← X − Y)	SUB (X ← Y − Z)
SUB	SUB M	MUL (X ← X × Y)	MUL (X ← Y × Z)
MUL	MUL M	DIV (X ← X/Y)	DIV (X ← Y/Z)
DIV	DIV M		

10.7. Considere un computador hipotético con un repertorio de instrucciones formado por solo dos instrucciones de n bits. El primer bit especifica el código de operación, y los bits restantes direccionan una de las 2^{n-1} palabras de n bits de la memoria principal. Las dos instrucciones son:

SUBS X Resta al acumulador el contenido de la posición X, y memoriza el resultado en la posición X y en el acumulador.

JUMP X Coloca la dirección X en el contador de programa.

Una palabra de memoria principal puede contener o una instrucción o un número binario en notación de complemento a dos. Demuestre que este repertorio de instrucciones es razonablemente completo especificando cómo se podrían programar las siguientes operaciones:

a) Transferencia de datos: posición X al acumulador, acumulador a la posición X.
b) Suma: suma el contenido de la posición X al acumulador.
c) Salto condicional.
d) Operación lógica OR.
e) Operaciones de E/S.

10.8. Muchos repertorios de instrucciones incluyen la instrucción NOOP (no operación), que no tiene otro efecto sobre el estado del procesador que incrementar el contador de programa. Sugiera algunos usos de esta instrucción.

10.9. En la Sección 10.4 indicamos que tanto el desplazamiento aritmético como el lógico a la izquierda corresponden a una multiplicación por dos cuando no se produce desbordamiento; y si se produce desbordamiento ambas operaciones producen resultados diferentes, pero el desplazamiento aritmético mantiene el signo del número. Compruebe que se cumple esta afirmación con números de cinco bits en complemento a dos.

10.10. De qué forma se redondean los números al efectuar desplazamientos aritméticos a la derecha (por ejemplo, redondeo hacia $+\infty$, redondeo hacia $-\infty$, hacia cero, alejándose de cero)?

10.11. Suponga que un procesador utiliza una pila para gestionar las llamadas a procedimientos y los retornos. ¿Puede eliminarse el contador de programa, utilizando la propia cabecera de la pila como contador de programa?

10.12. La arquitectura Pentium incluye una instrucción llamada Ajuste Decimal tras la Suma (DAA). DAA realiza la siguiente secuencia de operaciones:

```
if ((AL AND 0FH) > 9) OR (AF = 1) then
    AL ← AL + 6;
    AF ← 1;
else
    AF ← 0;
endif;
if (AL > 9FH) OR (CF = 1) then
    AL ← AL + 60H;
    CF ← 1;
else
    AF ← 0;
endif.
```

H indica hexadecimal. AL es un registro de ocho bits que retiene el resultado de la suma de dos enteros sin signo de ocho bits. AF es un indicador que se pone a uno si hay un acarreo del bit 3 al 4 como resultado de la suma. CF es un indicador que se activa si hay un acarreo del bit 7 al 8. Explique la función realizada por la instrucción DAA.

10.13. La instrucción del Pentium *Compare* (CMP: comparar) resta el operando origen del operando de destino; actualiza los indicadores de estado (C, P, A, Z, S, O) pero no afecta a ninguno de los operandos. La instrucción CMP puede usarse para determinar si el operando de destino es mayor, igual o menor que el operando origen.

a) Suponga que los dos operandos se tratan como enteros sin signo. Explique qué indicadores de estado son relevantes para determinar el tamaño relativo de los dos enteros, y qué valores corresponden a mayor que, igual a, y menor que.

b) Suponga que los dos operandos se tratan como enteros con signo en complemento a dos. Explique qué indicadores de estado son relevantes para determinar el tamaño relativo de los dos enteros, y qué valores corresponden a mayor que, igual a, y menor que.

c) La instrucción CMP puede estar seguida de una instrucción de salto condicional *Jump* (Jcc) o por una SETcc, donde cc hace referencia a una de las 16 condiciones listadas en la Tabla 10.11. Demuestre que son correctas las condiciones comprobadas para el caso de una comparación de números con signo.

10.14. Suponga que quisiéramos aplicar la instrucción CMP del Pentium a operandos de 32 bits que contienen números en un formato de coma flotante. ¿Qué requisitos de los siguientes apartados deben cumplirse para que los resultados sean correctos?

a) La posición relativa de los campos de signo, exponente y parte significativa.

b) La representación del valor cero.

c) La representación del exponente.

d) ¿Cumple estos requisitos el formato del IEEE? Justifique la respuesta.

10.15. La mayoría de los repertorios de instrucciones de microprocesadores incluyen una instrucción que comprueba una condición y, si se cumple, pone a uno un operando destino. Algunos ejemplos son la SETcc del Pentium, la Scc del Motorola MC68000, y la Scond del National 32000.

a) Hay algunas diferencias entre estas instrucciones:

- SETcc y Scc actúan sobre un único byte, mientras que Scond lo hace sobre operandos de un byte, una palabra o una palabra doble.
- SETcc y Scond ponen el operando al valor entero 1 si se cumple la condición, y a cero si no. Scc pone todos los bits del byte a uno o a cero respectivamente.

Compare las ventajas y desventajas relativas de estas diferencias.

b) Ninguna de estas instrucciones afecta a los indicadores de condición, y se requiere por tanto una comprobación explícita del resultado de la instrucción para determinar su valor. Discuta si los códigos de condición debieran activarse o no con esta instrucción.

c) Una sencilla sentencia IF, como la IF a > b THEN, puede implementarse utilizando un método de representación numérica; es decir, haciendo evidente el valor booleano, en contraposición al método de flujo del control, que representa el valor de una expresión booleana por un punto alcanzado en el programa. Un compilador podría implementar IF a > b THEN con el siguiente código del 80 × 86:

```
        SUB    CX,CX    ;pone el registro CX a 0
        MOV    AX,B     ;transfiere el contenido de la posición B al registro AX
        CMP    AX,A     ;compara el contenido del registro AX y de la posición A
        JLE    TEST     ;saltar si A ( B
        INC    CX       ;suma 1 al contenido del registro CX
TEST           JCXZ     OUT      ;salta si el contenido de CX es 0
THEN
OUT
```

El resultado de (A > B) es un valor booleano almacenado en un registro y disponible después, fuera del contexto del flujo del código mostrado más arriba. Es conveniente utilizar para ello el registro CX, porque muchos de los códigos de operación de bifurcación y de bucle incorporan la comprobación de CX.

Muestre una implementación alternativa utilizando la instrucción SETcc, que ahorre memoria y tiempo de ejecución (*Nota:* aparte de las SETcc no son necesarias otras instrucciones adicionales del 80 × 86).

d) Considere ahora la sentencia en lenguaje de alto nivel:

$$A := (B > C) \text{ OR } (D = F)$$

Un compilador podría generar el siguiente código:

```
        MOV    EAX, B   ;transfiere el contenido de la posición B al registro EAX
        CMP    EAX, C   ;compara el contenido del registro EAX y de la posición C
        MOV    BL, 0    ;0 representa falso
        JLE    N1       ;salta si B ≤ C
        MOV    BL, 1    ;1 representa falso
N1             MOV      EAX, D
        CMP    EAX, F
        MOV    BH, 0
        JNE    N2
        MOV    BH, 1
N2             OR       BL, BH
```

muestre una implementación alternativa utilizando la instrucción SETcc que ahorre memoria y tiempo de ejecución.

10.16. Suponga que dos registros contienen los siguientes valores hexadecimales: AB0890C2. 4598EE50. Indique cuál es el resultado de sumarlos utilizando instrucciones MMX:

a) Para byte empaquetado.

b) Para palabra empaquetada.

Suponga que no se usa aritmética con saturación.

10.17. El Apéndice 10A puntualiza que no hay instrucciones específicas para la pila en un repertorio de instrucciones cuando esta va a ser utilizada por el procesador solo para manejar procedimientos. ¿Cómo puede el procesador utilizar la pila para cualquier fin sin instrucciones especificas de gestión de la pila?

10.18. Convierta las siguientes fórmulas de notación polaca inversa a infija:

a) AB + C + D ×

b) AB/CD/ +

c) ABCDE + × ×/

d) ABCDE + F/ + G − H/× +

10.19. Convierta las siguientes fórmulas de infija a polaca inversa:

a) A + B + C + D + E

b) (A + B) × (C + D) + E

c) (A × B) + (C × D) + E

d) (A − B) × (((C − D × E)/F)/G) × H

10.20. Convierta la expresión A + B - C a notación postfija utilizando el algoritmo de Dijkstra. Muestre los pasos seguidos. ¿Es el resultado equivalente a (A + B) − C, o a A + (B − C)? ¿Importa?

10.21. Utilizando el algoritmo para conversión de notación infija a postfija definido en el Apéndice 10A, muestre los pasos involucrados en la conversión a postfija de la expresión de la Figura 10.15. Utilice una representación similar a la empleada en la Figura 10.17.

10.22. Muestre el cálculo de la expresión de la Figura 10.17, empleando una representación similar a la Figura 10.16.

10.23. Redibuje el patrón extremo menor de la Figura 10.18 de manera que los bytes aparezcan como si estuvieran numerados siguiendo el patrón extremo mayor. Es decir, muestre la memoria en filas de 64 bits, con los bytes enumerados de izquierda a derecha y de arriba a abajo.

10.24. Utilizando el formato de la Figura 10.18, dibuje los patrones extremo menor y extremo mayor para las siguientes estructuras de datos, y comente los resultados.

a)
```
struct {
      double i;    //0x1112131415161718
} s1;
```

b)
```
struc {
      int i;       //0x11121314
      int j;       //0x15161718
} s2;
```

c)
```
struct {
      short i;     //0x1112
      short j;     //0x1314
      short k;     //0x1516
      short l;     //0x1718
} s3;
```

10.25. Las especificaciones de la arquitectura PowerPC no definen cómo debe el procesador implementar el modo extremo menor (*little endian*). Indique solo cómo debe ver un procesador la memoria cuando se trabaja en modo extremo menor. Cuando se convierten estructuras de datos de extremo mayor (*big endian*) a extremo menor, los procesadores son libres de implementar un mecanismo de intercambio de bytes verdadero o de utilizar algún mecanismo de modificación de direcciones. Los procesadores

PowerPC actuales son por defecto máquinas extremo mayor y utilizan la modificación de direcciones para tratar los datos como extremo menor.

Considere la estructura definida en la Figura 10.18. El patrón de la parte inferior derecha de la figura muestra la estructura *s* como la vería el procesador. De hecho, si la estructura *s* se compila en modo extremo menor, su patrón en memoria es el que se muestra en la Figura 10.12. Explique la correspondiente asignación, describa una forma fácil para implementarla y discuta la efectividad de esta aproximación.

10.26. Escriba un pequeño programa para determinar el tipo de modo de extremo de una máquina e informe del resultado. Ejecute el programa en un computador al que tenga acceso y muestre el resultado.

10.27. El procesador MIPS puede programarse para que funcione en modo extremo mayor o bien en modo extremo menor. Considere la instrucción LBU (*Load Byte Unsigned,* cargar byte sin signo), que carga un byte de memoria en los ocho bits menos significativos de un registro y rellena con ceros los 24 bits restantes del registro. La descripción de LBU se da en el manual de referencia del MIPS empleando un lenguaje de transferencia entre registros:

mem $\leftarrow$ CargarMemoria(…)
byte $\leftarrow$ $\text{DireccionVirtual}_{1..0}$
if CONDICION **then**
  GRP[rt] $\leftarrow 0^{24}$ _ $\text{mem}_{31-8} \times {}_{_\text{byte}...24-8} \times {}_{\text{byte}}$
else
  GRP[rt] $\leftarrow 0^{24}$ _ $\text{mem}_{7+8} \times {}_{\text{byte}...8} \times {}_{\text{byte}}$
endif

en donde byte hace referencia a los dos bits menos significativos de la dirección efectiva, y mem hace referencia al valor cargado de memoria. En el manual, en lugar de la palabra CONDICION se emplea una de las dos siguientes: *big endian, little endian.*
¿Cuál de ellas se utiliza?

10.28. Muchos procesadores, aunque no todos, utilizan una ordenación extremo mayor o extremo menor de los bits dentro de un byte, que es consistente con el orden extremo mayor o extremo menor de los bytes dentro de un escalar multibyte. Consideremos el Motorota 68030, que emplea la ordenación de bytes extremo mayor. La documentación del 68030 relativa a los formatos es bastante confusa. El manual de usuario explica que la ordenación de los bits es la opuesta del orden que tienen en los enteros. La mayoría de los cálculos con campos o posiciones de bits operan con una modalidad de «extremo», pero hay unas cuantas operaciones que lo hacen en el orden contrario. La siguiente descripción, extraída del manual de usuario, detalla la mayoría de las operaciones con campos de bits mencionadas:

Asignación de direcciones
little endian

Dirección de byte								
					11	**12**	**13**	**14**
00	00	01	02	03	04	05	06	07
	21	**22**	**23**	**24**	**25**	**26**	**27**	**28**
08	08	09	0A	0B	0C	0D	0E	0F
	'D'	**'C'**	**'B'**	**'A'**	**31**	**32**	**33**	**34**
10	10	11	12	13	14	15	16	17
			51	**52**		**'G'**	**'F'**	**'E'**
18	18	19	1A	1B	1C	1D	1E	1F
					61	**62**	**63**	**64**
20	20	21	22	23	24	25	26	27

Figura 10.12. Estructura *s* extremo menor (*little endian*) en la memoria del PowerPC.

> Un operando bit se especifica mediante una dirección base que selecciona un byte de memoria (el byte base), y un número de bit que selecciona un bit dentro de dicho byte. El bit más significativo es el bit siete. Un operando «campo de bits» se especifica mediante (1) una dirección base que selecciona un byte de memoria; (2) un desplazamiento del campo de bit que indica el bit más a la izquierda (base) del campo de bit en relación al bit más significativo del byte base; y (3) la longitud del campo de bits, que determina cuántos bits a la derecha del bit base hay en el campo de bits. El bit más significativo del byte base tiene un desplazamiento de campo de bit 0, y el bit menos significativo del byte base tiene el desplazamiento (sesgo) de campo de bit 7.

De acuerdo con la anterior descripción, ¿estas instrucciones utilizan una ordenación extremo mayor o extremo menor?

APÉNDICE 10A. PILAS

PILAS

Una *pila* es un conjunto ordenado de elementos, en el que solo uno de ellos es accesible en un instante dado. El punto de acceso se denomina *cabecera* de la pila. El número de elementos en la pila, o *longitud* de la pila, es variable. Solo se pueden añadir o eliminar elementos en la cabecera de la pila. Por esta razón, una pila también se denomina *lista de apilamiento*[3] o *lista último-en-entrar-primero-en-salir* (LIFO: *Last-In-First-Out*).

La Figura 10.13 ilustra las operaciones básicas con la pila. Comenzamos en un instante de tiempo en el que la pila contiene algunos elementos. Una operación PUSH (apilar) añade un nuevo

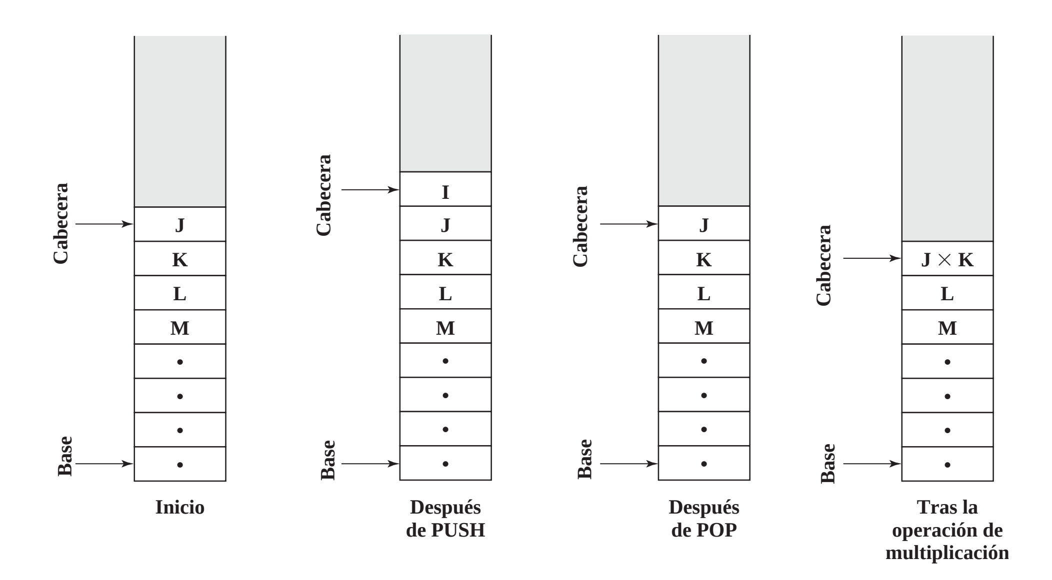


Figura 10.13. Operaciones básicas con la pila.

[3] Una descripción oportuna sería la de «ubicación en la cabecera de la lista», porque los elementos existentes de la lista no se cambian de posición de memoria, sino que cada nuevo elemento se añade en la siguiente dirección de memoria disponible.

elemento en la cabecera de la pila. Una operación POP (desapilar) elimina el elemento de la cabecera de la pila. En ambos casos, la cabecera experimenta el cambio apropiado. Las operaciones binarias, que requieren de dos operandos (por ejemplo, multiplicar, dividir, sumar, restar), utilizan dos elementos de la cabecera de la pila como operandos, desapilando ambos y apilando el resultado de nuevo en la pila. Las operaciones unarias, que requieren solo un operando (por ejemplo la NOT), utilizan el elemento de la cabecera de la pila. Todas estas operaciones se resumen en la Tabla 10.13.

IMPLEMENTACIÓN DE LA PILA

La pila es una estructura útil como parte de la implementación del procesador. Un posible uso, discutido en la Sección 10.4, es para la gestión de llamadas y retornos a/de procedimientos. Las pilas pueden ser también útiles para el programador. Un ejemplo es la evaluación de expresiones, discutida más adelante en esta sección.

La implementación de una pila depende en parte de sus usos potenciales. Si se desean ejecutar operaciones con la pila disponibles para el programador, el repertorio de instrucciones dispondría de operaciones orientadas al manejo de la pila, incluyendo PUSH, POP y operaciones que utilicen uno o dos elementos de la cabecera de la pila como operandos. Ya que todas estas operaciones hacen referencia a una misma posición, la cabecera de la pila, la dirección del operando u operandos está implícita y no necesita incluirse en la instrucción. Son, pues, instrucciones con cero direcciones a las que nos referimos en la Sección 10.1.

Si el mecanismo de pila va a ser utilizado solo por el procesador, con usos tales como el manejo de procedimientos, en el repertorio de instrucciones no se contemplarían instrucciones orientadas al uso de la pila. En cualquier caso, la implementación de una pila requiere que exista un cierto conjunto de posiciones utilizadas para almacenar los elementos de la pila. Una aproximación típica se ilustra en la Figura 10.14a. En memoria principal (o en memoria virtual) se reserva un bloque de posiciones contiguas para la pila. La mayor parte del tiempo, el bloque está parcialmente lleno con elementos de la pila, y el resto del bloque está disponible para que crezca la pila.

Para un funcionamiento correcto se necesitan tres direcciones, normalmente memorizadas en registros del procesador:

- **Puntero de pila:** contiene la dirección del tope o cabecera de la pila. Si se añade o se elimina un elemento de la pila, el puntero se incrementa o decrementa para que contenga la dirección de la nueva cabecera de la pila.

Tabla 10.13. Operaciones orientadas al uso de la pila.

PUSH	Añade un nuevo elemento en la cabecera de la pila.
POP	Elimina el elemento de la cabecera de la pila.
Operación Unaria	Realiza una operación con el elemento de la cabecera de la pila. Sustituye el elemento de la cabecera con el resultado.
Operación Binaria	Realiza una operación con dos elementos de la cabecera de la pila. Elimina de la pila dichos elementos. Pone el resultado de la operación en la cabecera de la pila.

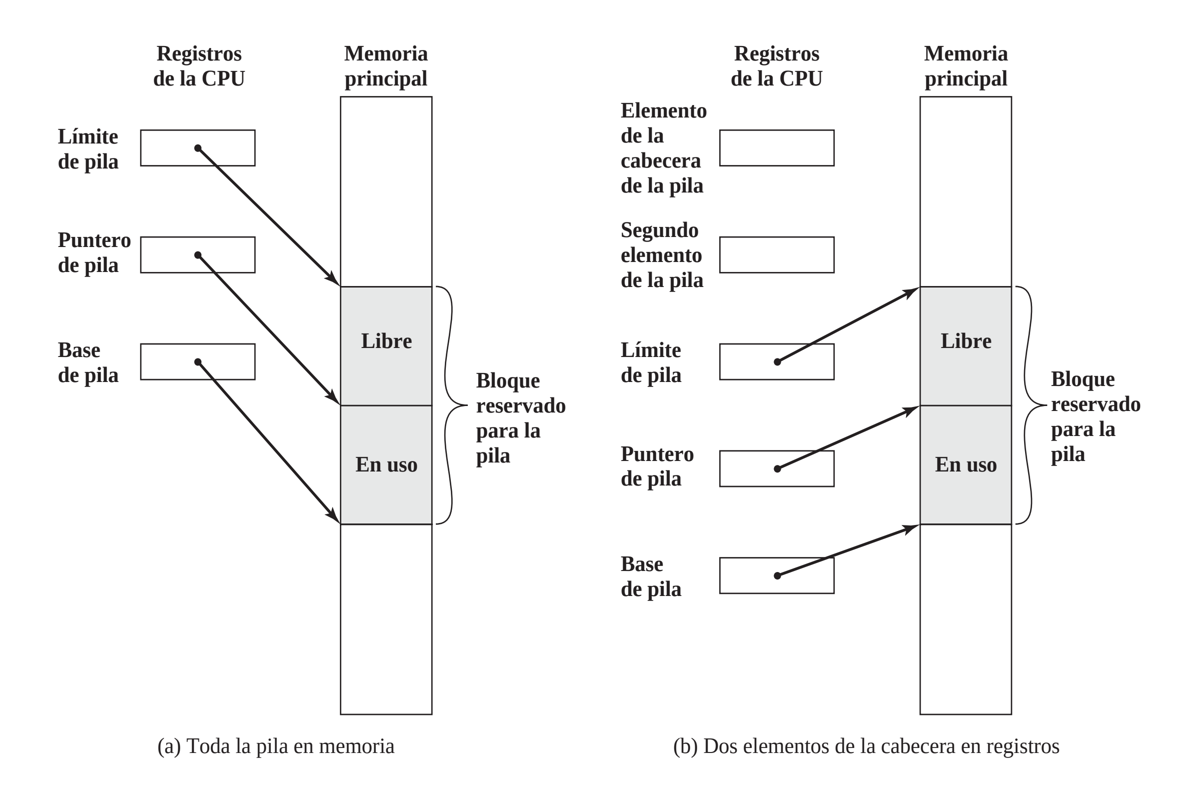


Figura 10.14. Organizaciones de pila usuales.

- **Base de la pila:** contiene la dirección base del bloque reservado para la pila. Si se intenta un POP cuando la pila está vacía, se informa de un error.
- **Límite de la pila:** contiene la dirección del otro extremo del bloque reservado. Si se intenta un PUSH cuando el bloque está utilizado en su totalidad, se informa de un error.

Tradicionalmente, y en la mayoría de las máquinas actuales, la base de la pila coincide con la dirección más alta del bloque reservado para la pila, mientras que el límite se corresponde con la dirección más baja. Por lo tanto, la pila crece desde direcciones más altas hacia más bajas.

Para acelerar las operaciones con la pila, los dos elementos de la cabecera se almacenan a menudo también en registros, como muestra la Figura 10.14b. En este caso, el puntero de pila contiene la dirección del tercer elemento de la pila.

EVALUACIÓN DE EXPRESIONES

Las fórmulas matemáticas se expresan normalmente en la notación conocida con el nombre de **infija**. En esta notación, el operador binario aparece entre los operandos (por ejemplo, a + b). En expresiones complejas se emplean paréntesis para determinar el orden de evaluación de las mismas. Por ejemplo, a + (b × c) producirá un resultado diferente que (a + b) × c. Para minimizar el número de paréntesis, las operaciones tienen una precedencia o prioridad implícita. Generalmente, la multiplicación tiene prioridad sobre la suma, de manera que a + b × c es equivalente a: a + (b × c).

Una técnica alternativa es la notación denominada **polaca inversa** o **postfija**. En esta notación, el operador se especifica después de los operandos. Por ejemplo,

a + b	sería a b +
a + (b × c)	sería a b c × +
(a + b) × c	sería a b + c ×

Observe que, por compleja que sea la expresión, no se requieren paréntesis en esta notación.

La ventaja de la notación postfija es que una expresión con este formato es fácil de evaluar usando una pila. Una expresión en notación postfija se recorre de izquierda a derecha y, para cada elemento de la expresión, se aplican las siguientes reglas:

1. Si el elemento es una variable o una constante, se introduce en la pila.
2. Si el elemento es un operador, se extraen de la cabecera de la pila los dos operandos, se realiza la operación, y se apila el resultado.

Tras recorrer la expresión completa, el resultado se encuentra en la cabecera de la pila.

La sencillez de este método lo hace muy apropiado para la evaluación de expresiones. Como consecuencia, muchos compiladores, partiendo de las expresiones descritas en lenguajes de alto nivel, las convierten a notación postfija, y entonces generan las instrucciones máquina a partir de esta notación. La Figura 10.15 muestra la secuencia de instrucciones máquina para evaluar f = (a − b)/(c + d × e) utilizando instrucciones orientadas al uso de la pila. La figura muestra también el uso de instrucciones con una y dos direcciones. Observe que, aunque en estos últimos casos no se han utilizado las instrucciones dedicadas a la pila, la notación postfija ha servido de guía para generar las instrucciones máquina. La secuencia de eventos producidos por el programa que emplea la pila se ilustra en la Figura 10.16.

El propio proceso de conversión de una expresión infija a otra postfija se lleva a cabo más fácilmente si se utiliza una pila. El siguiente algoritmo se debe a Dijkstra [DIJK63]. La expresión infija se

	Pila	**Registros generales**	**Registro único**
	Push a Push b Subtract Push c Push d Push e Multiply Add Divide Pop f	Load R1, a Subtract R1, b Load R2, d Multiply R2, e Add R2, c Divide R1, R2 Store R1, f	Load d Multiply e Add c Store f Load a Subtract b Divide f Store f
Número de instrucciones	10	7	8
Accesos a memoria	10 op + 6 d	7 op + 6 d	8 op + 8 d

Figura 10.15. Comparación de tres programas para calcular

$$f = \frac{a - b}{c + (d \times e)}$$

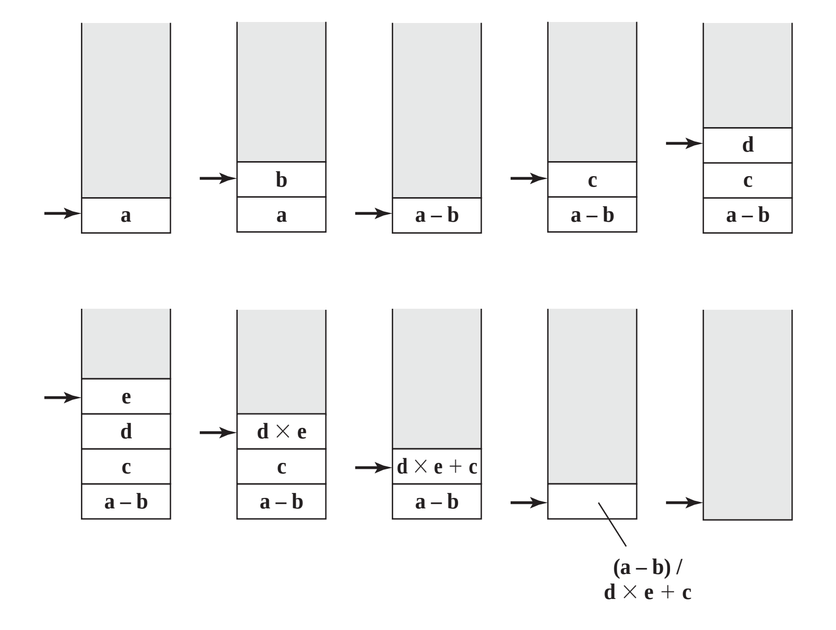


Figura 10.16. Utilización de la pila para calcular $f = (a - b)/(c + d \times e)$.

va recorriendo de izquierda a derecha, y se va generando a la vez la expresión postfija como salida. Los pasos del algoritmo son los siguientes:

1. Examinar el siguiente elemento de la linea de entrada.
2. Extraerlo como salida si se trata de un operando.
3. Si es una apertura de paréntesis, se introduce en la pila.
4. Si es un operador, entonces:
   - Si la cabecera de la pila contiene una apertura de paréntesis, apilar el operador.
   - Si tiene una prioridad más alta que el de la cabecera de la pila (la multiplicación y la división tienen mayor prioridad que la suma y la resta), apilar el operador.
   - Si tiene una prioridad menor o igual, efectuar un «pop» de la pila a la salida, y repetir el paso 4.
5. Si es un paréntesis de cierre, desapilar operadores hacia la salida hasta que se encuentre un paréntesis de apertura. Desapilar y descartar el paréntesis de apertura.
6. Si hay más entradas, ir al paso 1.
7. Si no hay más elementos de entrada, desapilar los restantes operadores.

La Figura 10.17 ilustra el uso de este algoritmo. Este ejemplo da al lector una idea de la potencia de los algoritmos basados en la pila.

Entrada	Salida	Pila (cabecera a la derecha)
A + B × C + (D + E) × F	vacía	vacía
+ B × C + (D + E) × F	A	vacía
B × C + (D + E) × F	A	+
× C + (D + E) × F	A B	+
C + (D + E) × F	A B	+ ×
+ (D + E) × F	A B C	+ ×
(D + E) × F	A B C × +	+
D + E) × F	A B C × +	+ (
+ E) × F	A B C × + D	+ (
E) × F	A B C × + D	+ (+
) × F	A B C × + D E	+ (+
× F	A B C × + D E +	+
F	A B C × + D E +	+ ×
vacía	A B C × + D E + F	+ ×
vacía	A B C × + D E + F × +	vacía

Figura 10.17. Conversión de una expresión de notación infija a postfija.

APÉNDICE 10B. ENDIAN: EXTREMO MENOR, EXTREMO MAYOR Y AMBOS EXTREMOS

Hay un aspecto curioso y molesto relacionado con la forma en que se referencian y se representan los bytes dentro de una palabra y los bits dentro de un byte. Veremos primero el problema del orden de los bytes, y después consideraremos el de los bits.

ORDEN DE LOS BYTES

El concepto de modo de extremo (*endian*) fue introducido por Cohen [COHE81]. En relación con los bytes, este concepto tiene que ver con la ordenación de los bytes en un escalar multibyte. Este tema se introduce mejor con un ejemplo. Suponga que tenemos el valor hexadecimal de 32 bits: 12345678, y que se almacena en una palabra de 32 bits de una memoria direccionable por bytes, en la posición de byte 184. El valor consta de cuatro bytes, con el contenido 78 para el byte menos significativo, y el byte más significativo con el valor 12 como contenido. Hay dos maneras de memorizar este valor:

Dirección	Valor
184	12
185	34
186	56
187	78

Dirección	Valor
184	78
185	56
186	34
187	12

La asignación de la izquierda almacena el byte más significativo en la dirección de byte con valor numérico más bajo; este modo se denomina extremo mayor (*big endian*), y es equivalente al orden de escritura de izquierda a derecha utilizado en las lenguas de las culturas occidentales. La asignación de la derecha almacena el byte menos significativo en la dirección con valor más bajo, y se conoce con el nombre de extremo menor (*little endian*), equivalente al orden de derecha a izquierda seguido para las operaciones aritméticas [4]. Para un valor escalar multi byte dado, la extremo menor y la extremo mayor son asignaciones inversas en términos del orden de los bytes.

El concepto de los *endians* surge cuando es necesario tratar una entidad multi byte como un único dato con una sola dirección, aun cuando esté compuesto de unidades direccionables más pequeñas. Algunos procesadores, tales como el Intel 80x86, el Pentium, VAX y Alpha, son máquinas extremo menor, mientras que otros, tales como el IBM System 370/390, el Motorola 680x0, Sun SPARC, y la mayoría de los RISC, son máquinas extremo mayor. Esto presenta problemas cuando se transfieren datos de una máquina con un tipo de *endian* a otra de distinto tipo, y cuando un programador intenta manipular bytes o bits individuales de un escalar multi byte.

La característica de *endian* se aplica solo a unidades de datos individuales. En cualquier máquina, elementos tales como los ficheros, las estructuras de datos, y las matrices, constan de múltiples unidades de datos, cada una de ella con la característica de *endian*. Así pues, la conversión de un bloque de memoria con un tipo de *endian* a otro, requiere conocer la estructura de los datos.

La Figura 10.18 ilustra cómo el tipo de *endian* determina el orden de direccionamiento y de los bytes. La estructura C de la parte superior contiene varios tipos de datos. El trazado de la memoria de la parte inferior izquierda es el resultado de la compilación de dicha estructura para una máquina extremo mayor, y el de la parte inferior derecha para una extremo menor. En ambos casos se muestra la memoria mediante una serie de filas de 64 bits. Para el caso extremo mayor, la memoria se representa de izquierda a derecha y de arriba a abajo, mientras que para el caso extremo menor se hace de derecha a izquierda y de arriba a abajo. Observe que estos trazados son arbitrarios. Cualquiera de los esquemas podría emplear un trazado de izquierda a derecha o de derecha a izquierda dentro de una fila; esto es algo que depende de la forma de hacer el trazado, y no de la asignación de memoria. De hecho, al mirar los manuales de programación de diversas máquinas, encontramos una confusa colección de formatos, incluso dentro de un mismo manual.

Podemos hacer varias observaciones acerca de esta estructura de datos:

- Cada elemento de datos tiene la misma dirección en ambos esquemas. Por ejemplo, la dirección de la palabra doble con el valor hexadecimal 2122232425262728, es 08.
- Dentro de un valor escalar multibyte dado, el orden de los bytes en la estructura extremo menores el inverso del de la estructura extremo mayor.
- El tipo de *endian* no afecta al orden de los elementos de datos dentro de una estructura. Así, en la palabra *c* de cuatro caracteres, los bytes se invierten, pero no en la matriz de bytes *d*, de siete caracteres. Por lo tanto, la dirección de cada elemento individual de d es la misma en ambas estructuras.

[4] Los términos *big endian* y *little endian* proceden de la Parte I, Capítulo 4 de *Los viajes de Gulliver,* de Jonathan Swift. Se refieren a una guerra religiosa entre dos grupos, uno que rompía los huevos por el extremo más grueso (*big endian*), y el otro que lo hacía por el extremo más estrecho (*little endian*).

```
struct{
  int     a;     //0x1112_1314                   palabra
  int     pad;   //
  double  b;     //0x2122_2324_2526_2728         doble palabra
  char*   c;     //0x3132_3334                   palabra
  char    d[7];  //'A'.'B','C','D','E','F','G'   matriz de bytes
  short   e;     //0x5152                        media palabra
  int     f;     //0x6162_6364                   palabra
} s;
```

Asignación de direcciones *big endian*

Dirección de byte								
	11	**12**	**13**	**14**				
00	00	01	02	03	04	05	06	07
	21	**22**	**23**	**24**	**25**	**26**	**27**	**28**
08	08	09	0A	0B	0C	0D	0E	0F
	31	**32**	**33**	**34**	**'A'**	**'B'**	**'C'**	**'D'**
10	10	11	12	13	14	15	16	17
	'E'	**'F'**	**'G'**		**51**	**52**		
18	18	19	1A	1B	1C	1D	1E	1F
	61	**62**	**63**	**64**				
20	20	21	22	23				

Asignación de direcciones *little endian*

								Direción de byte
				11	**12**	**13**	**14**	
07	06	05	04	03	02	01	00	00
21	**22**	**23**	**24**	**25**	**26**	**27**	**28**	
0F	0E	0D	0C	0B	0A	09	08	08
'D'	**'C'**	**'B'**	**'A'**	**31**	**32**	**33**	**34**	
17	16	15	14	13	12	11	10	10
		51	**52**		**'G'**	**'F'**	**'E'**	
1F	1E	1D	1C	1B	1A	19	18	18
				61	**62**	**63**	**64**	
				23	22	21	20	20

Figura 10.18. Ejemplo de estructura de datos en C, y sus mapas *endians*.

El efecto del tipo de *endian* se muestra quizás más claramente si consideramos la memoria como una matriz vertical de bytes, como se muestra en la Figura 10.19.

No existe un consenso general sobre qué tipo de *endian* es mejor [5]. Los siguientes puntos hacen preferible el estilo extremo mayor:

- **Ordenación de cadenas de caracteres:** un procesador extremo mayores más rápido en comparar cadenas de caracteres alineadas como enteros; la ALU de enteros puede comparar varios bytes en paralelo.
- **Volcados decimal/ASCII:** todos los valores pueden imprimirse de izquierda a derecha sin causar confusión.
- **Orden coherente:** los procesadores extremo mayor almacenan en el mismo orden los enteros y las cadenas de caracteres (el byte más significativo primero).

Los puntos siguientes favorecen al estilo extremo menor:

- Un procesador extremo mayor tiene que efectuar sumas cuando convierte una dirección entera de 32 bits a una de 16 bits al objeto de utilizar los bytes menos significativos.

[5] El profeta venerado por ambos grupos en las guerras de *Endians* de *Los Viajes de Gulliver* dijo: «Todos los creyentes convencidos deben romper los huevos por el extremo conveniente». Esto no nos resulta de gran ayuda.

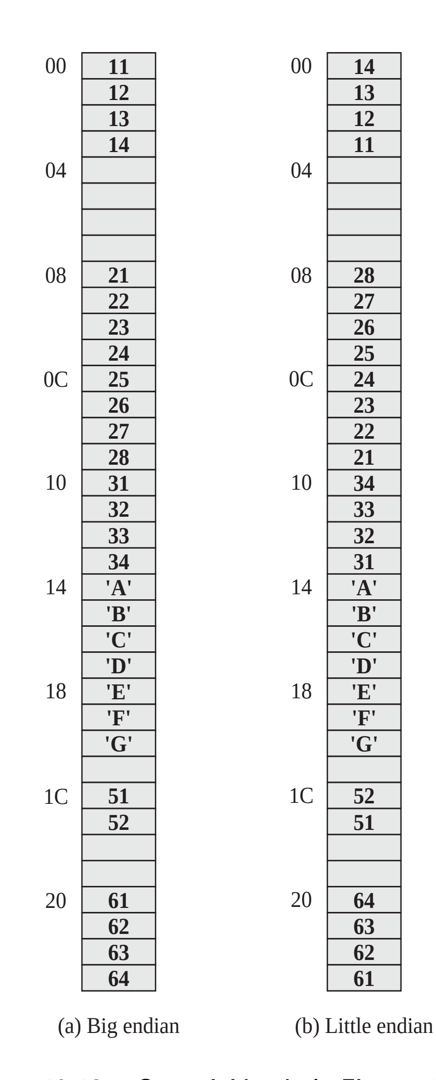


Figura 10.19. Otra visión de la Figura 10.18.

- Es más fácil realizar operaciones aritméticas de alta precisión con el estilo extremo menor; no hay que buscar primero el byte menos significativo y luego retroceder.

Las diferencias son poco significativas, y la elección del estilo de *endian* está más motivada por la necesidad de compatibilizar con máquinas anteriores que por otras causas.

El PowerPC es un procesador ambos extremos (*bi-endian*) que admite los dos modos extremo menor y extremo mayor. La arquitectura ambos extremos permite a los que desarrollan software elegir uno u otro modo cuando migran aplicaciones o sistemas operativos de otras máquinas. El sistema operativo establece el modo de *endian* en el que se ejecutan los procesos. Una vez seleccionado un modo, todas las transferencias a/desde memoria subsiguientes emplean ese modo. Para materializar esta facilidad hardware, el sistema operativo mantiene dos bits del registro de estado de la máquina (MSR) como parte del estado del proceso. Un bit especifica el modo de *endian* en que ejecuta el núcleo; el otro especifica el modo de operación actual del procesador. Así pues, el modo puede cambiarse de proceso a proceso.

ORDEN DE LOS BITS

Al considerar el orden de los bits dentro de un byte, inmediatamente nos planteamos dos cuestiones:

1. ¿Se cuenta el primer bit como bit número 0 o como número 1?
2. ¿El número de bit más bajo lo asignamos al bit menos significativo del byte (extremo menor) o al más significativo (extremo mayor)?

Estas cuestiones no se responden por igual para las distintas máquinas. Realmente, en algunas máquinas, las respuestas son diferentes según las circunstancias. Además, la elección entre extremo mayor o menor para la ordenación de bits dentro de un byte no es siempre coherente con la ordenación extremo mayor o menor de los bytes en un escalar multibyte. El programador necesita ser consciente de estos detalles cuando manipula bits individuales.

Otro aspecto a considerar es la transmisión de datos a través de una línea serie. Cuando se transmite un byte individual, ¿el sistema envía primero el bit más significativo o el menos significativo? El diseñador debe asegurarse de que los bits recibidos son tratados correctamente. Para una discusión sobre este tema, *véase* [JAME90].

CAPÍTULO 11

Repertorio de instrucciones: modos de direccionamiento y formatos

PUNTOS CLAVE

➤ La referencia a un operando en una instrucción contiene o bien su valor (inmediato) o una referencia a la dirección del operando. Los diversos repertorios de instrucciones utilizan una gran variedad de modos de direccionamiento. Estos modos incluyen el direccionamiento directo (la dirección del operando está en el campo de direcciones), indirecto (el campo de direcciones apunta a la posición que contiene la dirección del operando), a registro, indirecto con registro, y diversos tipos de desplazamiento en los que el valor de un registro se suma a un valor de dirección para producir la dirección del operando.

➤ El formato de instrucción define la forma de los distintos campos de la instrucción. El diseño del formato de instrucciones es una tarea difícil que debe considerar la longitud de las instrucciones, fija o variable, los números de bits asignados al código de operación y a cada referencia a operando, y la forma en que se determina el modo de direccionamiento.

En el Capítulo 10 nos hemos centrado en *qué* hace una instrucción. Concretamente, hemos examinado los tipos de operandos y de operaciones que pueden especificarse mediante instrucciones máquina. Este Capítulo 11 aborda la cuestión de *cómo* especificar los operandos y las operaciones de las instrucciones. Hay dos aspectos a tener en cuenta. El primero es cómo especificar la dirección de un operando; y el segundo, cómo se organizan los bits de una instrucción para definir las direcciones de los operandos y la operación que realiza dicha instrucción.

11.1. DIRECCIONAMIENTO

El campo o campos de direcciones en un formato de instrucción usual está bastante limitado. Sería deseable poder referenciar un rango elevado de posiciones de memoria principal o, en algunos sistemas, de memoria virtual. Para conseguir este objetivo se han empleado diversas técnicas de direccionamiento. Todas ellas implican algún compromiso entre el rango de direcciones y/o flexibilidad de direccionamiento de una parte y, por otra, el número de referencias a memoria y/o la complejidad de cálculo de las direcciones. En esta sección analizamos los modos de direccionamiento más comunes:

- Inmediato
- Directo
- Indirecto
- Registro
- Indirecto con registro
- Con desplazamiento
- Pila

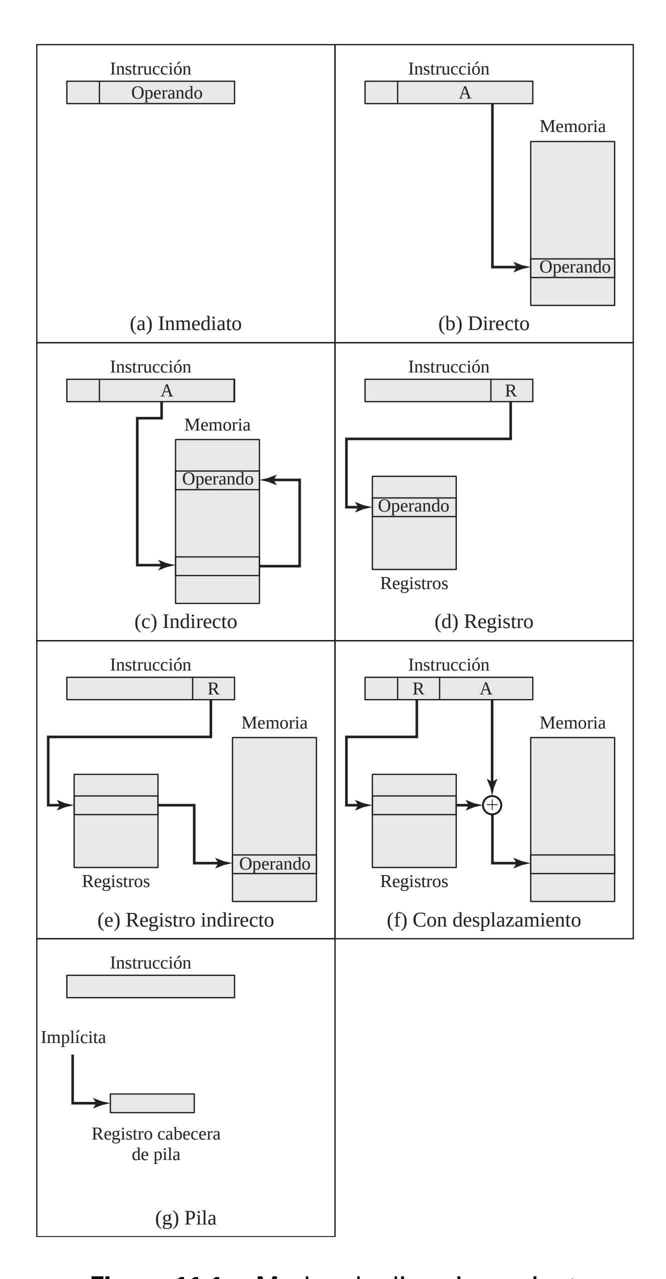


Figura 11.1. Modos de direccionamiento.

Estos modos se ilustran en la Figura 11.1. En esta sección utilizaremos la siguiente notación:

A = contenido de un campo de dirección en la instrucción

R = contenido de un campo de dirección en la instrucción que referencia a un registro

EA = dirección real (efectiva) de la posición que contiene el operando que se referencia

(X) = contenido de la posición de memoria X o del registro X

Tabla 11.1. Modos de direccionamiento básicos.

Modo	Algoritmo	Principal ventaja	Principal desventaja
Inmediato	Operando = A	No referencia a memoria	Operando de magnitud limitada
Directo	EA = A	Es sencillo	Espacio de direcciones limitado
Indirecto	EA = (A)	Espacio de direcciones grande	Referencias a memoria múltiples
Registro	EA = R	No referencia a memoria	Número limitado de registros
Indirecto con registro	EA = (R)	Espacio de direcciones grande	Referencia extra a memoria
Con desplazamiento	EA = A + (R)	Flexibilidad	Complejidad
Pila	EA = cabecera de la pila	No referencia a memoria	Aplicabilidad limitada

La Tabla 11.1 indica el cálculo de la dirección realizado para cada modo de direccionamiento.

Antes de comenzar esta discusión debemos hacer dos comentarios. El primero es que prácticamente todas las arquitecturas de computadores ofrecen más de uno de estos modos de direccionamiento. La cuestión que surge es cómo determina la unidad de control qué modo de direccionamiento se está empleando en cada instrucción. Se adoptan diversos enfoques alternativos. A menudo, codops diferentes emplearán modos de direccionamiento distintos. También, uno o más bits del formato de instrucción pueden utilizarse como *campo de modo*. El valor del campo de modo determina qué modo de direccionamiento va a utilizarse.

El segundo comentario se refiere a la dirección efectiva (EA, *effective address*). En un sistema sin memoria virtual, la *dirección efectiva* será o una dirección de memoria principal o un registro. En un sistema con memoria virtual, la dirección efectiva es una dirección virtual o un registro. La correspondencia real con una dirección física dependerá del mecanismo de paginación y no está visible al programador.

DIRECCIONAMIENTO INMEDIATO

La forma más sencilla de direccionamiento es el direccionamiento inmediato, en el que el operando está en realidad presente en la propia instrucción:

$$\text{Operando} = \text{A}$$

Este modo puede utilizarse para definir y utilizar constantes o para fijar valores iniciales de variables. Normalmente, el número se almacena en complemento a dos; el bit más a la izquierda del campo de operando se utiliza como bit de signo. Cuando el operando se carga en un registro de datos, el bit de signo se replica hacia la izquierda hasta la longitud total de la palabra de datos.

La ventaja del direccionamiento inmediato es que, una vez captada la instrucción, no se requiere una referencia a memoria para obtener el operando, ahorrándose pues un ciclo de memoria o de caché

en el ciclo de instrucción. La desventaja es que el tamaño del número está restringido a la longitud del campo de direcciones que, en la mayoría de los repertorios de instrucciones, es pequeño comparado con la longitud de palabra.

DIRECCIONAMIENTO DIRECTO

Una forma muy sencilla de direccionamiento es el direccionamiento directo, en el que el campo de direcciones contiene la dirección efectiva del operando:

$$EA = A$$

Esta técnica fue común en las primeras generaciones de computadores y se encuentra aún en diversos sistemas. Solo requiere una referencia a memoria y no necesita ningún cálculo especial. La limitación obvia es que proporciona un espacio de direcciones restringido.

DIRECCIONAMIENTO INDIRECTO

El problema del direccionamiento directo es que la longitud del campo de direcciones es normalmente menor que la longitud de palabra, limitando pues el rango de direcciones. Una solución es hacer que el campo de direcciones referencie la dirección de una palabra de memoria, la cual contenga la dirección completa del operando. Esto es lo que se denomina *direccionamiento indirecto*:

$$EA = (A)$$

Como se ha definido anteriormente, el paréntesis se interpreta como «contenido de». La ventaja obvia de esta aproximación es que para una longitud de palabra de N bits, se dispone ahora de un espacio de direcciones de 2^N. La desventaja es que la ejecución de la instrucción requiere dos referencias a memoria para captar el operando: una para captar su dirección y otra para obtener su valor.

Aunque el número de palabras que pueden ahora direccionarse es 2^N, el número de direcciones efectivas diferentes que pueden referenciarse en un instante dado está limitado a 2^K, donde K es la longitud del campo de direcciones. Normalmente, esto no supone una restricción severa, y puede superarse. En un entorno de memoria virtual, todas las posiciones de direcciones efectivas pueden confinarse a la página 0 de cualquier proceso. Ya que el campo de direcciones de una instrucción es pequeño, solo se podrán generar direcciones directas con valores bajos, las cuales aparecerán en la página 0 (la única restricción es que el tamaño de página debe ser mayor o igual que 2^K). Cuando un proceso está activo, habrá repetidas referencias a la página 0, haciendo que se quede en memoria real. Por tanto, una referencia indirecta a memoria implicará, como mucho, un fallo de página en lugar de dos.

Una variante de direccionamiento indirecto raramente utilizada es el direccionamiento multinivel o en cascada:

$$EA = (\ldots(A)\ldots)$$

En este caso, uno de los bits de una palabra de dirección es un indicador de «indirección» (I). Si el bit I es 0, la palabra contiene el valor de EA. Si el bit I es 1, entonces se invoca otro nivel de

«indirección». No parece que esta aproximación ofrezca ninguna ventaja particular, y su desventaja es que podrían requerirse dos o más referencias para captar un operando.

DIRECCIONAMIENTO DE REGISTROS

El direccionamiento de registros es similar al directo. La única diferencia es que el campo de direcciones referencia un registro en lugar de una dirección de memoria principal:

$$EA = R$$

Por ejemplo, si el campo de dirección de registro de la instrucción es 5, la dirección deseada es el registro R5, y el valor del operando es el contenido de R5. Normalmente, un campo de direcciones que referencia a registros consta de tres a cinco bits, de manera que pueden referenciarse un total de 8 a 32 registros de uso general.

Las ventajas del direccionamiento de registros son que (1) solo es necesario un campo pequeño de direcciones en la instrucción, y (2) no se requieren referencias a memoria. Como se comentó en el Capítulo 4, el tiempo de acceso a un registro interno del procesador es mucho menor que para la memoria principal. La desventaja del direccionamiento a registros es que el espacio de direcciones está muy limitado.

Si se hace un uso masivo del direccionamiento a registros en un repertorio de instrucciones, los registros del procesador se emplearán intensamente. Debido al número tan limitado de registros (en comparación con el número de posiciones de memoria principal), usarlos de esta manera tiene sentido solo si se emplean eficientemente. Si todo operando se carga de memoria principal a un registro, se opera con él una vez y se devuelve a memoria principal, resulta que se estaría añadiendo un paso intermedio improcedente. Si en lugar de esto, el operando del registro se mantiene en uso durante varias operaciones, se estaría consiguiendo un ahorro real. Un ejemplo son los resultados intermedios de un cálculo. En particular, suponga que se va a implementar en software el algoritmo de multiplicación en complemento a dos. La posición A del diagrama de flujo (Figura 9.12) se referencia muchas veces y debiera estar implementada en un registro en lugar de en una posición de memoria principal.

Depende del programador decidir qué valores deben mantenerse en registros y cuáles deben almacenarse en memoria principal. La mayoría de los procesadores modernos emplean múltiples registros de uso general, cargando al programador en lenguaje ensamblador (cuando desarrolla compiladores, por ejemplo) con la responsabilidad de conseguir una ejecución eficiente.

DIRECCIONAMIENTO INDIRECTO CON REGISTRO

Igual que el direccionamiento de registros es análogo al directo, el direccionamiento indirecto con registro es análogo al direccionamiento indirecto. En ambos casos, la diferencia estriba en si el campo de direcciones hace referencia a una posición de memoria o a un registro. Así, para el direccionamiento indirecto con registro:

$$EA = (R)$$

Las ventajas y limitaciones del direccionamiento indirecto con registro son básicamente las mismas que se tienen para el direccionamiento indirecto. En ambos casos, la limitación de espacio o rango de

direcciones del campo de direcciones, se supera haciendo que dicho campo referencie una posición de una palabra completa (un registro completo, en este caso) que contenga la dirección. Además, el direccionamiento indirecto con registro emplea una referencia menos a memoria que el direccionamiento indirecto.

DIRECCIONAMIENTO CON DESPLAZAMIENTO

Un modo muy potente de direccionamiento combina las posibilidades de los direccionamientos directo e indirecto con registro. Se conoce con distintos nombres dependiendo del contexto en que se emplee, pero el mecanismo básico es el mismo. Nosotros usaremos la denominación de *direccionamiento con desplazamiento:*

$$EA = A + (R)$$

El direccionamiento con desplazamiento requiere que las instrucciones tengan dos campos de direcciones, al menos uno de los cuales explícito. El valor contenido en uno de los campos de direcciones (valor = A) se utiliza directamente. El otro campo de direcciones, o una referencia implícita definida por el código de operación, se refiere a un registro cuyo contenido se suma a A para generar la dirección efectiva.

Describiremos tres de los usos más comunes del direccionamiento con desplazamiento:

- Desplazamiento relativo.
- Direccionamiento con registro base.
- Indexado.

Direccionamiento relativo. Para el direccionamiento relativo, también llamado direccionamiento relativo al PC, el registro referenciado implícitamente es el contador de programa (PC). Es decir, la dirección de instrucción actual se suma al campo de direcciones para producir el valor EA. Normalmente, el campo de direcciones se trata como número en complemento a dos para esta operación. En consecuencia, la dirección efectiva es un desplazamiento relativo a la dirección de la instrucción.

El desplazamiento relativo aprovecha el concepto de localidad discutido en los Capítulos 4 y 8. Si la mayoría de las referencias a memoria están relativamente cerca de la instrucción en ejecución, el uso del direccionamiento relativo ahorra bits de direcciones en la instrucción.

Direccionamiento con registro base. La interpretación del direccionamiento con registro-base es la siguiente: el registro referenciado contiene una dirección de memoria, y el campo de dirección contiene un desplazamiento (normalmente en representación entera sin signo) desde dicha dirección. La referencia a registro puede ser explícita o implícita.

El direccionamiento con registro base aprovecha también la localidad de las referencias a memoria. Es una forma conveniente de implementar la segmentación, que se estudió en el Capítulo 8. En algunas implementaciones se emplea un solo registro de base de segmento y es utilizado implícitamente. En otras, el programador puede seleccionar un registro para guardar la dirección de base de un

segmento, y la instrucción debe referenciarlo de manera explícita. En este último caso, si la longitud del campo de direcciones es K y el número de registros posibles es N, una instrucción puede referenciar una de entre N áreas de 2^K palabras.

Indexado. La interpretación usual del indexado es la siguiente: el campo de dirección referencia una dirección de memoria principal, y el registro referenciado contiene un desplazamiento positivo desde esa dirección. Obsérvese que este uso es justo el opuesto de la interpretación del direccionamiento con registro base. Por supuesto, hay algo más que una mera cuestión de interpretación de uso. Ya que en el indexado se considera que el campo de direcciones es una dirección de memoria, generalmente contiene más bits que un campo de direcciones de una instrucción comparable que emplee direccionamiento con registro base. También veremos que hay algunas mejoras del indexado que no serían tan útiles en el contexto de registros base. No obstante, el método para calcular EA es el mismo para ambos tipos de direccionamiento, y en ambos las referencias a los registros son a veces explícitas y a veces implícitas (para diferentes procesadores).

Un uso importante del indexado es como mecanismo eficiente para ejecutar operaciones iterativas. Considere por ejemplo una lista de números almacenados a partir de la posición A. Suponga que se quiere sumar 1 a cada elemento de la lista. Necesitamos captar cada valor, sumarle 1 y memorizar el resultado. La secuencia de direcciones efectivas necesarias es A, A + 1, A + 2..., hasta la última posición de la lista. Con el indexado, esto es fácil de hacer. El valor A se almacena en el campo de dirección de la instrucción, y el registro elegido, llamado *registro índice*, se inicializa a 0. Tras cada operación, el registro índice se incrementa en 1.

Dado que los registros índice se usan normalmente para tales tareas iterativas, es normal incrementarlos o decrementarlos tras cada referencia.

Ya que esta es una operación común, algunos sistemas la efectúan automáticamente como parte del ciclo de instrucción. Esto se denomina *autoindexado*. Si un registro concreto esta dedicado exclusivamente al indexado, el autoindexado puede ser invocado implícitamente y automáticamente. Si se emplean registros de uso general, la operación de autoindexado puede requerir de un bit de la instrucción que lo indique. El autoindexado con incremento puede describirse como sigue:

$$EA = A + (R)$$

$$(R) \leftarrow (R) + 1$$

Algunas máquinas disponen de direccionamiento tanto indirecto como indexado, y es posible emplear ambos en la misma instrucción. Hay dos posibilidades: el indexado se realiza bien antes, o bien después de la «indirección».

Si la indexación se realiza después de la indireccion se denomina *post-indexado*:

$$EA = (A) + (R)$$

Primero, el contenido del campo de direcciones se emplea para acceder a la posición de memoria que contiene una dirección directa. Esta dirección se indexa entonces mediante el valor del registro. Esta técnica es útil para acceder a uno de entre un número de bloques de datos con un formato fijo. Por ejemplo, en el Capítulo 8 se describió que el sistema operativo necesita emplear un bloque de control para cada proceso. Las operaciones que se realizan son las mismas independientemente del bloque

manipulado. Por lo tanto, las direcciones indicadas en las instrucciones que referencian el bloque podrían apuntar a una posición (valor = A) que contenga un puntero variable hacia el inicio del bloque de control de proceso. El registro índice contiene el desplazamiento dentro del bloque.

Con *preindexado*, la indexación se realiza antes de la «indirección»:

$$EA = (A + (R))$$

La dirección se calcula como en el caso de indexado simple. No obstante, en este caso, la dirección calculada no contiene al operando, sino la dirección del operando. Un ejemplo de uso de esta técnica es la construcción de tablas de bifurcación multirama. En un punto concreto de un programa puede haber una bifurcación a una de entre varias posiciones, en función de diversas condiciones. Puede establecerse una tabla de direcciones que comience en la posición A, y mediante indexado en esta tabla, encontrar la posición requerida.

Normalmente, un mismo repertorio de instrucciones no incluirá el preindexado y el postindexado simultáneamente.

DIRECCIONAMIENTO DE PILA

El último modo de direccionamiento que consideramos es el de pila. Como definimos en el Apéndice 10A, una pila es una matriz lineal de posiciones. A veces se le denomina *lista de apilamiento* o *cola último-en-entrar-primero-en-salir*. Una pila es un bloque de posiciones reservado. Los elementos se añaden en la cabecera de la pila de manera que, en cualquier instante, el bloque está parcialmente lleno. La pila tiene asociado un puntero cuyo valor es la dirección de la cabecera o tope de la pila. Alternativamente, los dos elementos de la cabecera de la pila pueden residir en registros del procesador, en cuyo caso el puntero de pila hace referencia al tercer elemento de la pila (Figura 10.14b). El puntero de pila se mantiene en un registro. Así, las referencias a posiciones de la pila en memoria son, de hecho, direcciones de acceso indirecto con registro.

El modo de direccionamiento de pila es una forma de direccionamiento implícito. Las instrucciones máquina no necesitan incluir una referencia a memoria sino que operan implícitamente con la cabecera de la pila.

11.2. MODOS DE DIRECCIONAMIENTO EN EL PENTIUM Y EL POWERPC

MODOS DE DIRECCIONAMIENTO DEL PENTIUM

Recordemos, de la Figura 8.21, que el mecanismo de traducción de direcciones del Pentium produce una dirección, denominada dirección virtual o efectiva, que es un desplazamiento dentro de un segmento. La suma de la dirección de comienzo del segmento y la dirección efectiva produce una dirección lineal. Si se está empleando paginación, esta dirección lineal debe pasar por un mecanismo de traducción de páginas para producir una dirección física. En lo que sigue ignoraremos este último paso, ya que es transparente para el repertorio de instrucciones y para el programador.

El Pentium está equipado con diversos modos de direccionamiento, ideados para permitir la ejecución eficiente de lenguajes de alto nivel. La Figura 11.2 indica el hardware involucrado. El segmento objeto de la referencia se determina mediante el registro de segmento. Hay seis registros de segmento; cuál de ellos es usado para una referencia concreta depende del contexto de ejecución y de la instrucción. Cada registro de segmento retiene la dirección de comienzo del correspondiente segmento. Asociado con cada registro de segmento visible para el usuario, hay un registro descriptor de segmentos (no visible al programador), que registra los derechos de acceso para el segmento, así como la dirección de comienzo y el límite (longitud) del segmento. Además, hay dos registros que pueden emplearse para construir una dirección: el registro base y el registro índice.

La Tabla 11.2 lista los doce modos de direccionamiento del Pentium. Consideremos cada uno de ellos por separado.

En el **modo inmediato**, el operando se incluye en la instrucción. El operando puede ser un byte, una palabra o una palabra doble de datos.

En el **modo de operando en registro,** el operando está situado en un registro. Para instrucciones de tipo general, tales como transferencias de datos, aritméticas, y lógicas, el operando puede ser uno de los registros generales de 32 bits (EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP), uno de los registros generales de 16 bits (AX, BX, CX, DX, SI, DI, SP, BP), o uno de los registros generales de ocho bits (AH, BH, CH, DH, AL, BL, CL, DL). Para operaciones en coma flotante, los operandos de

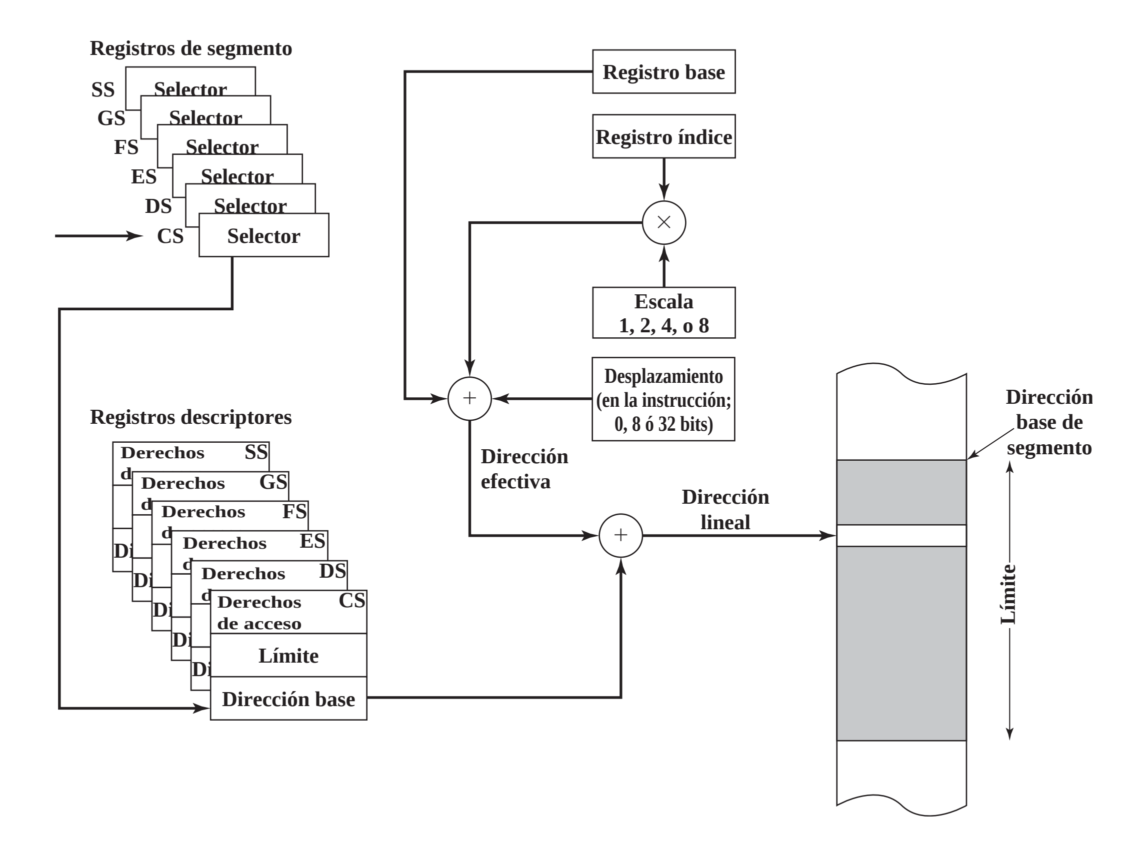


Figura 11.2. Cálculos en el modo de direccionamiento del Pentium.

Tabla 11.2. Modos de direccionamiento del pentium.

Modo	Algoritmo
Inmediato	Operando = A
Registro	LA = R
Con deplazamiento	LA = (SR) + A
Base	LA = (SR) + (B)
Base con desplazamiento	LA = (SR) + B + A
Índice escalado con desplazamiento	LA = (SR) + (I) × S + A
Base con índice y desplazamiento	LA = (SR) + (B) + (I) + A
Base con índice escalado y desplazamiento	LA = (SR) + (I) × S + (B) + A
Relativo	LA = (PC) + A

LA = dirección lineal
(X) = contenido de X
SR = registro de segmento
PC = contador de programa
A = contenido de un campo de dirección de la instrucción
R = registro
B = registro base
I = registro índice
S = factor de escala

64 bits se forman utilizando dos registros de 32 bits como una pareja. Hay también algunas instrucciones que hacen referencia a los registros de segmento (CS, DS, ES, SS, FS, GS).

Los modos de direccionamiento restantes referencian posiciones de memoria. La posición de memoria debe especificarse en términos del segmento que la contiene y el desplazamiento desde el comienzo del segmento. En algunos casos, el segmento se especifica de manera explícita; en otros, queda especificado por las reglas de asignación implícita de segmentos.

En el **modo de desplazamiento,** el desplazamiento del operando (la dirección efectiva en la Figura 11.2) está incluido, formando parte de la instrucción, como desplazamiento de 8, 16, o 32 bits.

Con segmentación, todas las direcciones dadas en instrucciones hacen referencia a desplazamientos dentro de segmentos.

El modo de direccionamiento con desplazamiento se puede encontrar en pocas máquinas ya que, como hemos mencionado antes, implica instrucciones largas. En el caso del Pentium, el valor de desplazamiento puede ser tan largo como 32 bits, haciendo que la instrucción tenga seis bytes. El direccionamiento con desplazamiento puede ser útil para referenciar variables globales.

Los modos de direccionamiento restantes son indirectos, en el sentido de que el campo de dirección de la instrucción indica al procesador dónde debe encontrar la dirección. El **modo base** especifica que uno de los registros de 8, 16, o 32 bits contiene la dirección efectiva. Esto es equivalente a lo que hemos denominado direccionamiento indirecto con registro.

En el **modo base con desplazamiento,** la instrucción incluye un desplazamiento que hay que sumar a un registro base, que puede ser cualquiera de los registros de uso general. Ejemplos de uso de este modo son:

- Utilización por un compilador para apuntar al comienzo de una zona de variables locales. Por ejemplo, el registro base podría apuntar al comienzo de un marco de pila, que contiene las variables locales para el procedimiento correspondiente.
- Utilización para indexar en una matriz cuando el tamaño de elemento no es 1, 2, 4 u 8 bytes, y no puede por tanto ser indexado utilizando un registro índice. En este caso, el desplazamiento apunta al comienzo de la matriz, y el registro base retiene los resultados del cálculo para determinar el desplazamiento de un elemento específico dentro de la matriz.
- Utilización para acceder a un campo de un registro. El registro base apunta al comienzo del registro, mientras que el desplazamiento indica la posición del campo.

En el **modo desplazamiento con índice «escalado»**, la instrucción incluye un desplazamiento a sumar a un registro, llamado en este caso registro índice. El registro índice puede ser cualquiera de los registros de uso general excepto el ESP, que normalmente se emplea para procesamiento con la pila. Para calcular la dirección efectiva, el contenido del registro índice se multiplica por un factor de escala 1, 2, 4 u 8, y se suma después a un desplazamiento. Este modo es muy conveniente para indexar matrices. Un factor de escala de dos puede usarse para una matriz de enteros de 16 bits. Un factor de escala cuatro para enteros de 32 bits o para números en coma flotante. Finalmente, un factor de escala ocho puede emplearse para una matriz de números en coma flotante de doble precisión.

El **modo base con índice y desplazamiento** suma los contenidos de los registros base e índice, y un desplazamiento, para formar la dirección efectiva. De nuevo, el registro base puede ser cualquier registro de uso general, y el registro índice puede ser cualquier registro de uso general excepto el ESP. Como ejemplo, este modo de direccionamiento podría emplearse para acceder a una matriz local en un marco de pila. Este modo puede usarse también para manejar una matriz bidimensional; en este caso, el desplazamiento apunta al inicio de la matriz, y cada registro gestiona una dimensión de la misma.

El **modo base con índice escalado y desplazamiento** suma el contenido de registro índice multiplicado por un factor de escala, con el contenido del registro base, y el desplazamiento. Esto es útil cuando una matriz está almacenada en un marco de pila; en este caso, los elementos de la matriz serían de 2, 4, u 8 bytes de longitud. Este modo permite también la indexación eficiente de una matriz bidimensional cuando los elementos de la misma tienen longitudes de 2, 4, u 8 bytes.

Finalmente, el **direccionamiento relativo** puede emplearse en instrucciones de transferencia del control (control de flujo). Se suma un desplazamiento al valor del contador de programa, que apunta a la instrucción siguiente. En este caso, el desplazamiento se trata como un byte, una palabra o una palabra doble numérica con signo, cuyo valor bien incrementa o decrementa la dirección contenida en el contador de programa.

MODOS DE DIRECCIONAMIENTO DEL POWERPC

Como la mayoría de las máquinas RISC, y en contraste con el Pentium y la mayoría de los CISC, el PowerPC emplea un conjunto de modos de direccionamiento sencillo y relativamente evidente. Como indica la Tabla 11.3, estos modos conviene clasificarlos con relación al tipo de instrucciones.

Direccionamiento de carga/memorización. El PowerPC proporciona dos modos de direccionamiento alternativos para instrucciones de carga/memorización (Figura 11.3). En el

Tabla 11.3. Modos de direccionamiento del PowerPC.

Modo	Algoritmo
	Direccionamiento para carga/memorización
Indirecto	EA = (BR) + D
Indirecto indexado	EA = (BR) + (IR)
	Direccionamiento de saltos
Absoluto	EA = I
Relativo	EA = (PC) + I
Indirecto	EA = (L/CR)
	Cálculos en coma fija
Registro	EA = GPR
Inmediato	Operando = I
	Cálculos en coma flotante
Registro	EA = FPR

EA = dirección efectiva
(X) = contenido de X
BR = registro base
IR = registro índice
L/CR = registro de enlace o de cuenta
GPR = registro de uso general
FPR = registro de coma flotante
D = desplazamiento
I = valor inmediato
PC = contador de programa

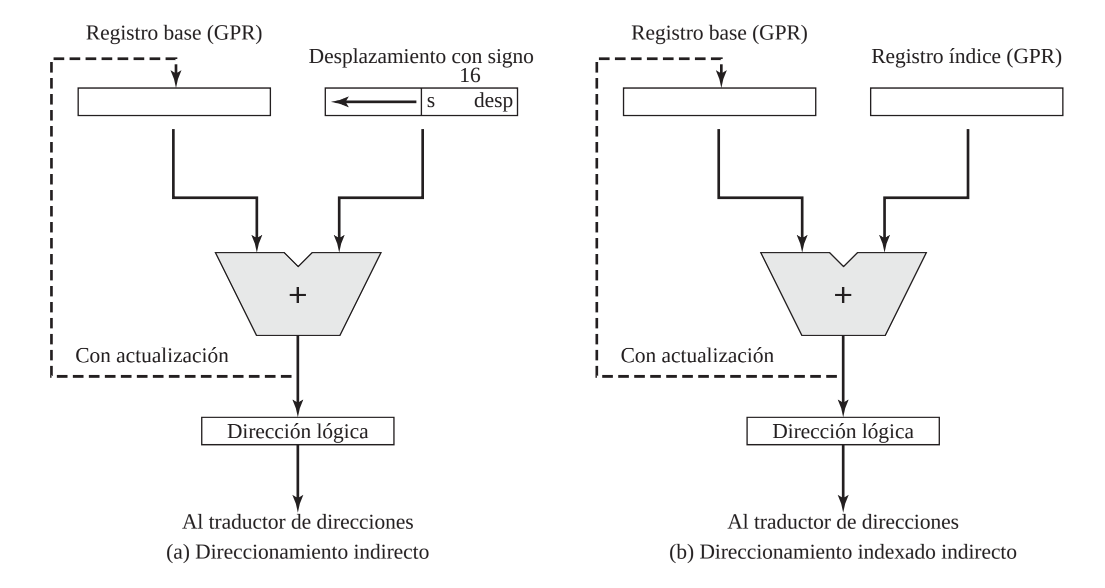


Figura 11.3. Modos de direccionamiento de operandos en memoria del PowerPC.

direccionamiento indirecto, la instrucción incluye un desplazamiento de 16 bits que se suma a un registro base, que puede ser alguno de los registros de uso general. Además, la instrucción puede especificar que la nueva dirección efectiva calculada se devuelva al registro base, actualizando su contenido actual. La opción de actualización es útil para el indexado progresivo de matrices en bucles.

La otra técnica de direccionamiento para las instrucciones de carga/memorización es el **direccionamiento indexado indirecto.** En este caso, la instrucción referencia un registro base y otro índice, pudiendo ambos ser cualesquiera de los registros de uso general. La dirección efectiva es la suma de los contenidos de estos dos registros. De nuevo, la opción de actualización hace que el registro base se actualice con la nueva dirección efectiva.

Direccionamiento de bifurcaciones. Se dispone de tres modos de direccionamiento para bifurcaciones. Si se emplea **direccionamiento absoluto** con instrucciones de salto incondicional, la dirección efectiva de la siguiente instrucción se obtiene a partir de un valor inmediato de 24 bits contenido en la instrucción. El valor de 24 bits se extiende hasta 32 bits añadiendo dos ceros a su extremo menos significativo (esto es posible ya que las instrucciones están cada 32 bits) y extendiendo (replicando) el signo. Para las instrucciones de salto condicional, la dirección efectiva de la siguiente instrucción se deduce de un valor inmediato de 16 bits incluido en la instrucción. Este valor se extiende a un valor de 32 bits añadiendo dos ceros a su extremo menos significativo y extendiendo el signo.

Con **direccionamiento relativo,** el valor inmediato de 24 bits (instrucciones de salto incondicional) o de 14 bits (instrucciones de salto condicional) se extiende como en los casos anteriores. El valor resultante se suma entonces al contador de programa a fin de determinar una posición relativa a la instrucción actual. El otro modo de direccionamiento de bifurcaciones condicionales es el **direccionamiento indirecto.** Este modo obtiene la dirección efectiva de la siguiente instrucción bien del registro de enlace o bien del registro de cuenta. En este caso, el registro de cuenta se usa para retener la dirección de la instrucción de bifurcación. Este registro puede también emplearse para mantener un contador para bucles, como explicamos con anterioridad.

Instrucciones aritméticas. En operaciones aritméticas con enteros, todos los operandos deben estar bien en registros o bien formar parte de la instrucción. Con direccionamiento de registros, un operando origen o destino se especifica como uno de los registros de uso general. Con direccionamiento inmediato, un operando origen aparece como cantidad de 16 bits, con signo, en la propia instrucción.

Para las operaciones aritméticas en coma flotante, todos los operandos se encuentran en registros de coma flotante; es decir, solo se utiliza direccionamiento de registros.

11.3. FORMATOS DE INSTRUCCIONES

Un formato de instrucciones define la descripción en bits de una instrucción, en términos de las distintas partes o campos que la componen. Un formato de instrucciones debe incluir un código de operación (codop) e, implícita o explícitamente, cero o más operandos. Cada operando explícito se referencia utilizando uno de los modos de direccionamiento descritos en la Sección 11.1. El formato debe, implícita o explícitamente, indicar el modo de direccionamiento para cada operando. En la mayoría de los repertorios de instrucciones se emplean más de un formato de instrucción.

El diseño de un formato de instrucción es una labor compleja, habiéndose implementando una variedad muy amplia de diseños. En esta sección, examinamos los aspectos clave de diseño, analizando brevemente algunos diseños que servirán de ilustración, y después examinaremos con detalle las soluciones adoptadas en el Pentium y en el PowerPC.

LONGITUD DE INSTRUCCIÓN

El aspecto de diseño más básico a considerar en el formato es la longitud o tamaño de la instrucción. Esta decisión afecta, y se ve afectada por, el tamaño de la memoria, su organización, la estructura de buses, la complejidad del procesador y la velocidad del procesador. Esta decisión define la riqueza y flexibilidad de la máquina desde el punto de vista del programador en lenguaje ensamblador.

El compromiso más obvio está entre el deseo de disponer de un repertorio de instrucciones máquina potente y la necesidad de ahorrar espacio. El programador desea más codops, más operandos, más modos de direccionamiento y mayor rango de direcciones. Más codops y más operandos facilitan el trabajo del programador, ya que puede redactar programas más cortos para resolver las mismas tareas. De manera similar, más modos de direccionamiento dan más flexibilidad al programador para implementar ciertas funciones, tales como la gestión de tablas y las bifurcaciones multi-rama. Y, por supuesto, con el aumento de tamaño de la memoria principal y el uso creciente de la memoria virtual, los programadores demandan poder direccionar rangos de memoria grandes. Todo esto (codops, operandos, modos de direccionamiento y rango de direcciones) requiere de bits y empuja hacia longitudes de instrucción mayores. Pero una longitud de instrucción mayor puede ser improcedente. Una instrucción de 64 bits ocupa el doble de espacio que una de 32 bits pero probablemente no es el doble de útil.

Aparte de este compromiso básico, hay otras consideraciones a tener en cuenta. Debiera cumplirse o bien que el tamaño de la instrucción fuera igual al tamaño de las transferencias a memoria (en un sistema basado en un bus, igual al tamaño del bus de datos), o bien que uno fuera un múltiplo del otro. En caso contrario, no conseguiremos un número íntegro de instrucciones durante un ciclo de captación. Una cuestión relacionada es la velocidad de transferencia de la memoria. Esta velocidad no ha seguido el mismo aumento que la velocidad de los procesadores. Por ello, la memoria puede convertirse en un cuello de botella si el procesador puede ejecutar las instrucciones más rápido que lo que tarda en captarlas. Una solución a este problema es el uso de memoria caché (*véase* la Sección 4.3); otra es utilizar instrucciones más cortas. De nuevo, las instrucciones de 16 bits pueden captarse el doble de rápido que las de 32 bits, pero probablemente no pueden ejecutarse el doble de rápido.

Una característica aparentemente irrelevante pero sin embargo importante es que la longitud de la instrucción debiera ser un múltiplo de la longitud de un carácter, que normalmente es ocho bits, y de la longitud de los números en coma fija. Para verlo necesitamos hacer uso de un término desafortunadamente mal definido, la *palabra* [FRAI83]. La longitud de palabra de memoria es, en cierto sentido, la unidad «natural» de organización. El tamaño de una palabra define normalmente el tamaño de los números en coma fija (usualmente ambos coinciden). El tamaño de palabra suele también coincidir, o al menos está directamente relacionado, con el tamaño de las transferencias a memoria. Ya que una forma común de datos es el carácter, sería deseable que una palabra almacenara un número entero de caracteres. Si no, se perderían bits en cada palabra cuando se almacenan múltiples caracteres, o habría algunos caracteres partidos entre dos palabras. La importancia de este punto es tal que IBM, cuando introdujo el Sistema/360 y quiso emplear caracteres de ocho bits, tuvo que adoptar la decisión drástica de pasar de la arquitectura de 36 bits de las máquinas científicas de las series 700/7000 a una arquitectura de 32 bits.

ASIGNACIÓN DE LOS BITS

Hemos visto algunos de los factores que influyen en la decisión de la longitud del formato de instrucción. Un aspecto igualmente difícil es cómo asignar los bits en dicho formato. Los compromisos a la hora de decidir son en este caso complejos.

Para una longitud de instrucción dada, existe claramente un compromiso entre el número de codops y la capacidad de direccionamiento. Un mayor número de codops obviamente implica más bits en el campo de codop. Esto reduce, para un formato de instrucción de una longitud dada, el número de bits disponibles para direccionamiento. Hay un refinamiento interesante al respecto, consistente en el uso de codops de longitud variable. En esta aproximación, existe una longitud mínima de codop pero, para algunos de ellos, se pueden especificar operaciones adicionales utilizando más bits de la instrucción. En una instrucción de longitud fija, esto deja menos bits para direccionamiento. Así pues, esta característica se emplea en aquellas instrucciones que requieren menos operandos y/o menor capacidad de direccionamiento.

Los siguientes factores, relacionados entre sí, afectan a la definición del uso dado a los bits de direccionamiento.

- **Número de modos de direccionamiento:** un modo de direccionamiento puede a veces indicarse de manera implícita. Por ejemplo, ciertos codops podrían siempre hacer referencia a indexación. En otros casos, los modos de direccionamiento deben ser explícitos, requiriéndose uno o más bits de modo.
- **Número de operandos:** hemos visto que menos direcciones pueden hacer que los programas sean más largos y difíciles (*véase* como ejemplo la Figura 10.3). Las instrucciones típicas de las máquinas actuales permiten dos operandos. Cada dirección de operando podría requerir su propio indicador de modo dentro de la instrucción, o el uso del indicador de modo podría estar limitado a solo uno de los campos de direcciones.
- **Registros frente a memoria:** una máquina debe disponer de registros para traer los datos al procesador a fin de procesarlos. En el caso de un solo registro visible para el usuario (denominado usualmente acumulador) la dirección del operando está implícita y no consume bits de la instrucción. Sin embargo, la programación con un único registro es engorrosa y requiere muchas instrucciones. Incluso con varios registros, solo se necesitan unos pocos bits para especificar el registro. Diversos estudios indican que es aconsejable disponer de 8 a 32 registros visibles para el usuario [LUND77, HUCK83]. La mayoría de las arquitecturas contemporáneas disponen de al menos 32 registros.
- **Número de conjuntos de registros:** varias máquinas tienen un conjunto de registros de uso general, que contiene 8 o 16 registros. Estos registros pueden emplearse para guardar datos y para almacenar direcciones para direccionamiento con desplazamiento. La tendencia actual ha sido pasar de un solo banco de registros de uso general a un grupo de dos o más conjuntos especializados (por ejemplo, para datos y para desplazamientos). Una ventaja de este enfoque es que, para un número dado de registros, una partición funcional de estos requiere menos bits de la instrucción. Por ejemplo, con dos conjuntos de ocho registros, solo se necesitan 3 bits para identificar un registro; el codop determina de forma implícita qué conjunto de registros se está referenciando.
- **Rango de direcciones:** para referencias a memoria, el rango de direcciones que puede utilizarse está relacionado con el número de bits de direccionamiento. Dado que esto impone una limitación severa, raramente se emplea direccionamiento directo. En direccionamiento con desplazamiento, el rango se amplía al definido por la longitud del registro de direcciones. Incluso así, es aún conveniente permitir desplazamientos bastante más largos que los del registro de direcciones, y esto requiere de un número relativamente grande de bits de direcciones en la instrucción.

- **Granularidad de las direcciones:** para direcciones que hacen referencia a memoria en lugar de registros, otro factor es la granularidad del direccionamiento. En un sistema con palabras de 16 o 32 bits, una dirección puede referenciar una palabra o un byte, según elija el diseñador. El direccionamiento por bytes es conveniente para manipular caracteres pero requiere, para un tamaño de memoria dado, de más bits de direcciones.

Por lo tanto, el diseñador se enfrenta con una gran cantidad de factores a tener en cuenta y sopesar. No está claro cuán críticas son las distintas opciones. Como ejemplo, citamos un estudio [CRAG79] que compara varios enfoques al formato de instrucciones, incluyendo el uso de una pila, registros de uso general, un acumulador y aproximaciones solo memoria-a-registro. Haciendo uso de un conjunto coherente de suposiciones, no se observan diferencias significativas en espacio de código o en tiempo de ejecución.

Veamos brevemente cómo dos diseños de máquinas históricas sopesan los distintos factores anteriores.

PDP-8. Uno de los diseños de instrucciones más sencillos para un computador de uso general fue el del PDP-8 [BELL78b]. El PDP-8 utiliza instrucciones de doce bits y opera con palabras de doce bits. Hay un solo registro de uso general, el acumulador.

A pesar de lo limitado de este diseño, el direccionamiento es bastante flexible. Cada referencia a memoria consta de siete bits, más dos modificadores de un bit. La memoria se divide en páginas de longitud fija, cada una con $2^7 = 128$ palabras. El cálculo de direcciones se basa en las referencias a la página cero o a la página actual (página que contiene la instrucción) según el valor del bit de página. El segundo bit modificador indica si se va a usar direccionamiento directo o indirecto. Estos dos modificadores pueden utilizarse conjuntamente, de tal manera que una dirección indirecta será una dirección de doce bits contenida en una palabra de la página cero o de la página actual. Además, ocho palabras dedicadas de la página cero son «registros» de autoindexado. Cuando se hace una referencia indirecta a una de estas posiciones, tiene lugar un preindexado.

La Figura 11.4 muestra el formato de instrucción del PDP-8. Hay un codop de tres bits y tres tipos de instrucciones. Para los codops 0 a 5, el formato consiste en una instrucción con una sola referencia a memoria, incluyendo un bit de página y un bit de indirección. Así pues, hay solo seis operaciones básicas. Para ampliar el grupo de operaciones, el codop 7 define una referencia a registro o *microinstrucción*. En este formato, los bits restantes se emplean para codificar operaciones adicionales. En general, cada bit define una operación específica (por ejemplo, poner el acumulador a cero), y estos bits pueden combinarse en una sola instrucción. La estrategia de las microinstrucciones la empleó DEC ya con el PDP-1, y es, en cierto sentido, un precursor de las máquinas microprogramadas actuales, que se tratarán en la Parte Cuatro del libro. El codop seis es la operación de E/S; se emplean seis bits para seleccionar uno de entre 64 dispositivos, y tres bits especifican una orden particular de E/S.

El formato de instrucción del PDP-8 es bastante eficiente. Permite direccionamiento indirecto, direccionamiento con desplazamiento, e indexado. Con el uso de la ampliación de codop, dispone de un total de 35 instrucciones. Debido a lo limitado de una longitud de instrucción de solo doce bits, los diseñadores difícilmente lo podrían haber hecho mejor.

PDP-10. En fuerte contraste con el repertorio de instrucciones del PDP-8 está el del PDP-10. El PDP-10 se diseñó para ser un sistema de tiempo compartido de gran escala, haciendo hincapié en que fuera fácil de programar, incluso a costa de hardware adicional.

Instrucciones con referencia a memoria

Codop			D/I	Z/C	Desplazamiento	
0		2	3	4	5	11

Instrucciones de entrada/salida

1	1	0	Dispositivo		Codop	
0		2	3	8	9	11

Instrucciones con referencia a registro

Microinstrucciones del grupo 1

1	1	1	0	CLA	CLL	CMA	CML	RAR	RAL	BSW	IAC
0	1	2	3	4	5	6	7	8	9	10	11

Microinstrucciones del grupo 2

1	1	1	0	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11

Microinstrucciones del grupo 3

1	1	1	0	CLA	MQA	0	MQL	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11

D/I = Direccionamiento directo/indirecto
Z/C = Página 0 ó página actual
CLA = Complementar acumulador
CLL = Complementar enlace
CMA = Complemento acumulador
CML = Complemento enlace
RAR = Rotar a derecha acumulador
RAL = Rotar a izquierda acumulador
BSW = Intercambiar Byte
IAC = Incrementar acumulador
SMA = Salto si acumulador negativo
SZA = Salto si acumulador cero
SNL = Salto si enlacedistinto de cero
RSS = Invertir sentido del salto
OSR = OR con registro conmutador
HLT = Parar
MQA = Multiplicador / cociente en el acumulador
MQL = Cargar multiplicador / cociente

Figura 11.4. Formatos de instrucciones del PDP-8.

Algunos de los principios de diseño que se emplearon al definir el repertorio de instrucciones fueron [BELL78c]:

- **Ortogonalidad:** es un principio que hace que dos variables sean independientes entre sí. En el contexto de los repertorios de instrucciones, este término indica que otros elementos de una instrucción son independientes de (no están determinados por) el codop. Los diseñadores del PDP-10 utilizan este término para describir el hecho de que una dirección se calcula siempre de la misma manera, independientemente del codop. En esto difiere de muchas máquinas, en las que el modo de direccionamiento a veces depende implícitamente del operador que se está usando.
- **Complitud:** cada tipo de datos aritméticos (enteros, en coma fija, reales) debiera disponer de un conjunto completo e idéntico de operaciones.
- **Direccionamiento directo:** el direccionamiento mediante base más desplazamiento, que responsabiliza al programador de la organización de la memoria, fue evitado en favor del direccionamiento directo.

Cada uno de los principios anteriores es un avance hacia la meta global de una programación fácil.

El PDP-10 tiene una longitud de palabra de 36 bits y una longitud de instrucción de 36 bits. El formato fijo de instrucción se muestra en la Figura 11.5. El codop ocupa nueve bits, permitiendo hasta 512 operaciones. De hecho, se define un total de 365 instrucciones diferentes. La mayoría de las

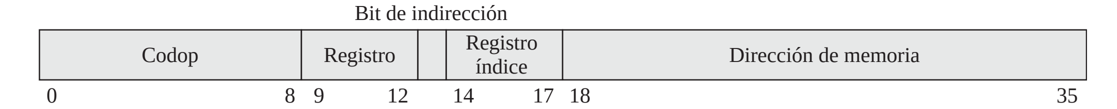


Figura 11.5. Formato de instrucción del PDP-10.

instrucciones incluyen dos direcciones, una de las cuales especifica uno de los 16 registros de uso general. Esta referencia a operando ocupa pues cuatro bits. La otra referencia a operando comienza con un campo de direcciones de 18 bits. Este puede utilizarse como operando inmediato o como una dirección de memoria. En el segundo caso, se permite direccionamiento tanto indirecto como indexado. Los mismos registros de uso general se emplean también como registros índice.

Una longitud de instrucción de 36 bits es realmente un lujo. No hay que hacer nada especial para conseguir aumentar el número de codops; un campo de código de operación de nueve bits es más que suficiente. El direccionamiento es también obvio. Un campo de dirección de 18 bits hace deseable el direccionamiento directo. Para tamaños de memoria superiores a 2^{18}, se dispone del direccionamiento indirecto. Para facilidad del programador, se dispone de indexado para manejar tablas y programas iterativos. También, con un campo de operando de 18 bits, resulta atractivo el direccionamiento inmediato.

El diseño de repertorio de instrucciones del PDP-10 cumple los objetivos antes enumerados [LUND77]. Este repertorio facilita el trabajo del programador a costa de una utilización ineficiente del espacio. Esto fue una decisión consciente de los diseñadores y no puede pues calificarse como un diseño poco elaborado.

INSTRUCCIONES DE LONGITUD VARIABLE

Los ejemplos vistos hasta ahora hacen uso de una única longitud de instrucción fija, e implícitamente hemos discutido compromisos adoptados en ese contexto. No obstante, los diseñadores pueden optar por utilizar varios formatos de instrucción de longitudes diferentes. Esta táctica hace fácil proporcionar un amplio repertorio de codops de longitud variable. El direccionamiento puede ser más flexible, con varias combinaciones de referencias a registros y a memoria, así como de modos de direccionamiento. Con instrucciones de longitud variable, estas múltiples variaciones pueden proporcionarse de manera eficiente y compacta.

El precio a pagar por las instrucciones de longitud variable es el aumento de complejidad del procesador. La disminución del precio del hardware, el uso de microprogramación (discutida en la Parte Cuatro) y un aumento general en el conocimiento de los principios de diseño de procesadores, contribuyen todos a hacer que el precio a pagar mencionado sea leve.

El uso de instrucciones de longitud variable no elimina el deseo de que todas las longitudes de instrucciones estén directamente relacionadas con la longitud de palabra. Ya que el procesador no conoce la longitud de la siguiente instrucción a captar, una estrategia usual es captar un número de bytes o de palabras igual, al menos, al tamaño de la instrucción más larga. Esto significa que a veces se captan varias instrucciones. Sin embargo, como veremos en el Capítulo 12, esta es siempre una buena a seguir.

PDP-11. El PDP-11 se diseñó para proporcionar un repertorio de instrucciones flexible dentro de las limitaciones de un minicomputador de 16 bits [BELL70].

El PDP-11 emplea un conjunto de ocho registros de uso general de 16 bits. Dos de estos registros tienen una función adicional: uno se emplea como puntero de pila en operaciones específicas con la pila, y el otro se emplea como contador de programa, que contiene la dirección de la siguiente instrucción.

La Figura 11.6 muestra los formatos de instrucciones del PDP-11. Se usan trece formatos diferentes, compaginando instrucciones de ninguna, una y dos direcciones. La longitud del codop puede variar de cuatro a 16 bits. Para las referencias a registros se emplean seis bits. Tres bits identifican el registro, y los otros tres restantes indican el modo de direccionamiento. El PDP-11 está dotado de una rica variedad de modos de direccionamiento. Una ventaja de asociar el modo de direccionamiento con el operando, en lugar de con el codop, como en ocasiones se hace, es que todos los modos de direccionamiento pueden emplearse con cualquier codop. Como hemos mencionado con anterioridad, esta independencia se denomina *ortogonalidad.*

Las instrucciones del PDP-11 tienen normalmente la longitud de una palabra (16 bits). En algunas instrucciones se añaden una o dos direcciones de memoria, así que hay instrucciones de 32 y de 48 bits formando parte del repertorio. Esto permite una flexibilidad de direccionamiento adicional.

El repertorio de instrucciones del PDP-11 y su capacidad de direccionamiento son complejos. Esto aumenta tanto el coste del hardware como la complejidad de programación. La ventaja es que pueden desarrollarse programas más eficientes o compactos.

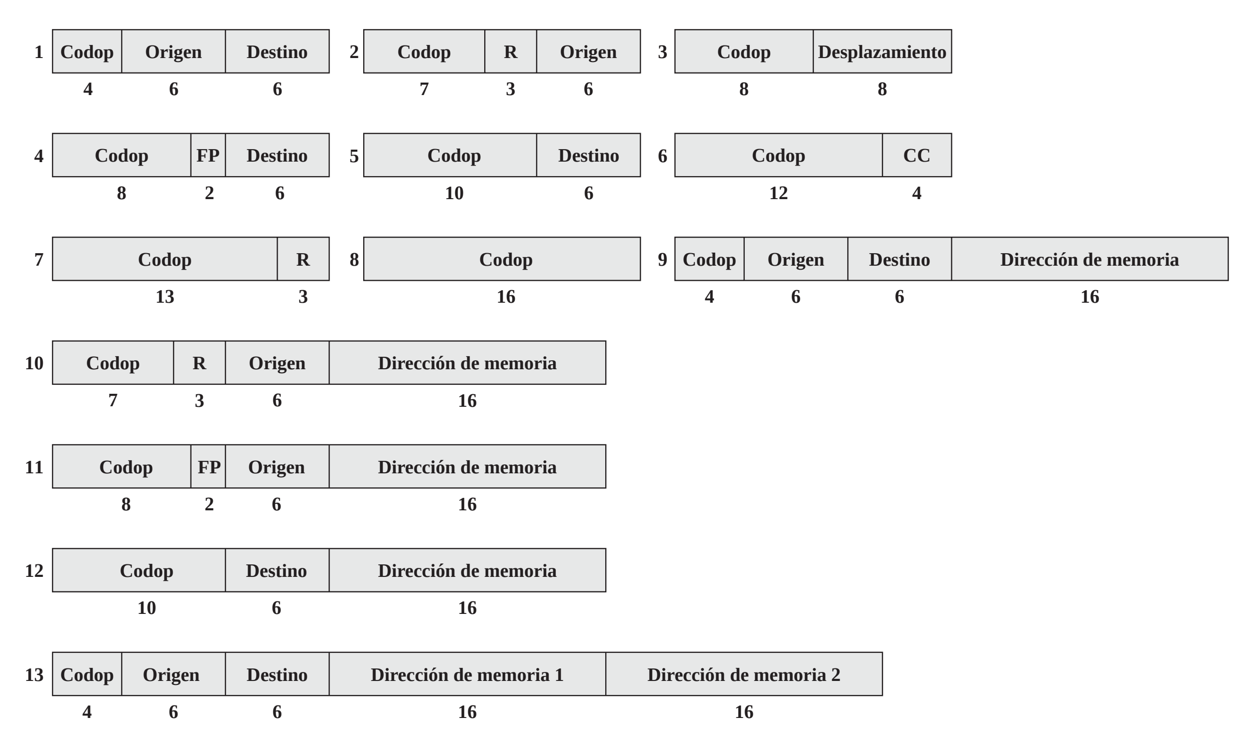


Figura 11.6. Formatos de instrucciones utilizados en el PDP-11.

VAX. La mayoría de las arquitecturas proporcionan un número relativamente pequeño de formatos de instrucciones. Esto puede causar dos problemas al programador. En primer lugar, el modo de direccionamiento y el codop no son ortogonales. Por ejemplo, para una operación dada, un operando debe proceder de un registro y otro de memoria, o ambos de registros, etc. En segundo lugar, solo puede contemplarse un número limitado de operandos: normalmente hasta dos o tres. Dado que algunas operaciones requieren más operandos, deben emplearse estrategias que permitan conseguir el resultado deseado mediante dos o más instrucciones.

Para evitar estos problemas, se siguieron dos criterios al diseñar el formato de instrucción del VAX [STRE78]:

1. Todas las instrucciones debían tener el número «natural» de operandos.

2. Todos los operandos debían tener la misma generalidad de especificación.

El resultado es un formato de instrucción muy variable. Una instrucción consta de un codop de uno o dos bytes, seguido de una especificación de cero a seis operandos, que depende del codop en cuestión. La longitud mínima de instrucción es un byte, pudiéndose construir instrucciones de hasta 37 bytes. La Figura 11.7 muestra algunos ejemplos.

Una instrucción del VAX comienza por un codop de 1 byte. Este es suficiente para representar la mayoría de las instrucciones del VAX. No obstante, como dispone de más de trescientas instrucciones

Formato hexadecimal	Explicación	Notación de ensamblador y descripción
8 bits 0 5	Codop de RSB	RSB Retorno de subrutina
D 4 5 9	Codop de CLRL Registro R9	CLRL R9 Limpiar el registro R9
B 0 C 4 6 4 0 1 A B 1 9	Codop de MOVW Modo de desplazamiento de palabra, registro R4 356 en hexadecimal Modo de desplazamiento de byte, registro R11 25 en hexadecimal	MOVW 356(R4), 25(R11) Transfiere una palabra la dirección: 356 más el contenido de R4, a la dirección: 25 más el contenido de R11
C 1 0 5 5 0 4 2 D F (desplazamiento)	Codop ADDL3 Literal corto 5 Modo registro R0 Prefijo de índice R2 Palabra indirecta relativa (desplazamiento desde el PC) Cantidad de desplazamiento desde el PC relativo a la posición A	ADDL3 #5, R0, @A[R2] Suma 5 al entero de 32 bits contenido en RO y almacena el resultado en la posición cuya dirección es la suma de A y de 4 veces el contenido de R2

Figura 11.7. Ejemplos de instrucciones del VAX.

diferentes, ocho bits no son suficientes. Los códigos hexadecimales FD y FF indican un codop ampliado: el codop real lo especifica un segundo byte.

El resto de la instrucción consta de hasta seis especificadores de operandos. Un especificador de operando ocupa como mínimo un byte, en el que los cuatro bits de la izquierda especifican el modo de direccionamiento. La única excepción a esta regla es el modo literal, que es identificado por el patrón 00 en los dos bits más a la izquierda, quedando espacio para un literal de seis bits. Debido a esta excepción, son un total de doce los modos de direccionamiento que se pueden especificar.

Un especificador de operando consiste a menudo en un solo byte, en el que los cuatro bits de la derecha especifican uno de entre los 16 registros de uso general. La longitud del especificador de operando puede ampliarse en una de las dos formas siguientes: en primer lugar, un valor constante de uno o más bytes puede seguir inmediatamente al primer byte del especificador de operando. Un ejemplo es el modo de desplazamiento, en el que se emplea un desplazamiento de 8, 16, o 32 bits. En segundo lugar, puede utilizarse un modo indexado de direccionamiento. En este caso, el primer byte del especificador de operando está formado por el código de modo de direccionamiento de cuatro bits 0100 y un identificador de registro índice de cuatro bits. El resto del especificador de operando contiene la especificación de la dirección base, que puede a su vez ocupar uno o más bytes.

Puede que el lector se pregunte, como también lo hizo el autor, qué clase de instrucción puede requerir seis operandos. Sorprendentemente, el VAX dispone de varias de estas instrucciones. Consideremos:

ADDP6 OP1, OP2, OP3, OP4, OP5, OP6

Esta instrucción suma dos números decimales empaquetados. OP1 y OP2 especifican la longitud y la dirección de inicio de una de las cadenas decimales; OP3 y OP4 especifican la segunda cadena. Las dos cadenas se suman, y el resultado es memorizado como cadena decimal cuya longitud y posición inicial vienen dadas por OP5 y OP6.

El repertorio de instrucciones del VAX ofrece una gran variedad de operaciones y de modos de direccionamiento. Esto proporciona al programador una herramienta muy potente y flexible para el desarrollo de programas. También, en teoría, debiera producir compilaciones eficientes en lenguaje máquina de programas en lenguajes de alto nivel y, en general, obtener un uso eficiente y efectivo de los recursos del procesador. El precio a pagar por estos beneficios es un incremento en la complejidad del procesador, en comparación con un procesador que posea un repertorio de instrucciones más reducido y de formato más sencillo.

Volveremos sobre estas cuestiones en el Capítulo 13, donde examinaremos el caso de repertorios de instrucciones muy sencillos.

11.4. FORMATOS DE INSTRUCCIONES DEL PENTIUM Y DEL POWERPC

FORMATOS DE INSTRUCCIÓN DEL PENTIUM

El Pentium está equipado con varios formatos de instrucciones. De los elementos descritos anteriormente, solo el campo codop está siempre presente. La Figura 11.8 ilustra el formato de instrucción

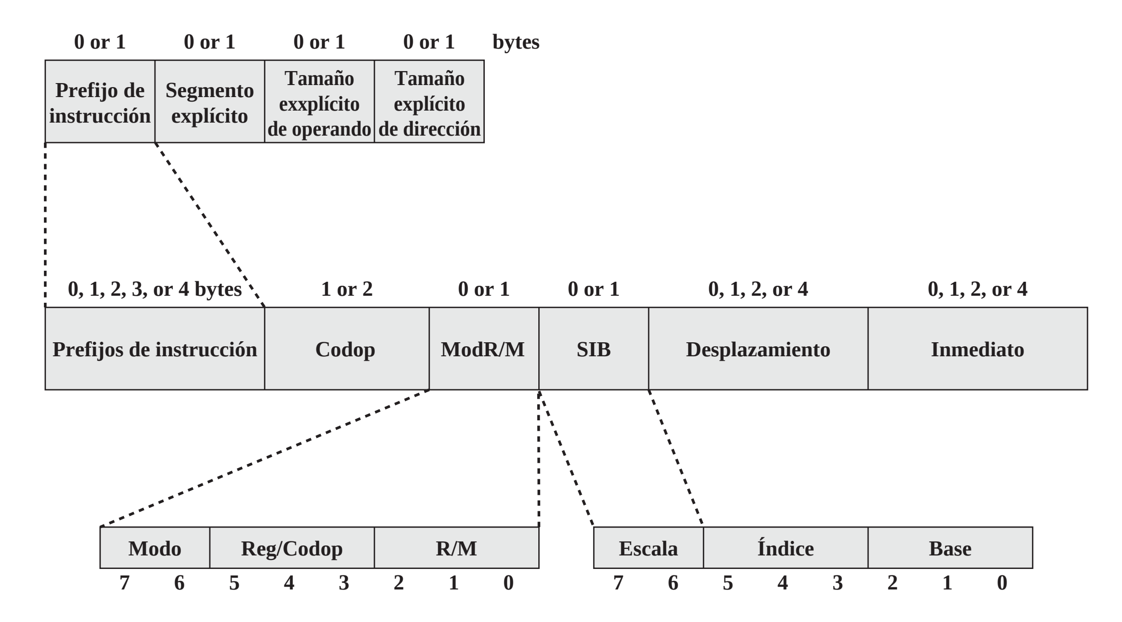


Figura 11.8. Formato de instrucciones del Pentium.

general. Las instrucciones se componen de entre cero y cuatro prefijos de instrucción opcionales, un codop de uno o dos bytes, una especificación de dirección opcional, que consta del byte {ModR/m» y del byte de índice de escala (*Scale Index*), un desplazamiento opcional y un campo inmediato opcional.

- **Prefijos de instrucción:** el prefijo de instrucción, si está presente, contiene el prefijo LOCK o uno de los prefijos de repetición. El prefijo LOCK se emplea para asegurar el uso exclusivo de la memoria compartida en entornos multiprocesador. Los prefijos de repetición indican que se repita una operación en una cadena, lo que permite al Pentium procesar cadenas mucho más rápido que con un bucle software habitual. Hay cinco prefijos de repetición diferentes: REP, REPE, REPZ, REPNE, y REPNZ. Cuando está presente el prefijo absoluto REP, la operación que se especifica en la instrucción se ejecuta repetidas veces con los elementos sucesivos de la cadena; el número de repeticiones se especifica en el registro CX. Un prefijo REP condicional hace que se repita la instrucción hasta que la cuenta de CX llegue a cero o hasta que se cumpla la condición.

- **Segmento explícito:** especifica de forma explícita qué registro de segmento deberá utilizar una instrucción, prevaleciendo sobre la selección de registro de segmento implícito generada por el Pentium para dicha instrucción.

- **Tamaño de la dirección:** el procesador puede direccionar memoria utilizando direcciones de 16 o de 32 bits. El tamaño de la dirección define el tamaño del desplazamiento indicado en las instrucciones y el tamaño de los desplazamientos de direcciones generados durante el cálculo de la dirección efectiva. Uno de estos tamaños se considera valor implícito, y el prefijo de tamaño de la dirección conmuta entre la generación de una dirección de 32 bits o una de 16 bits.

- **Tamaño de operando:** de manera similar, una instrucción tiene implícitamente un tamaño de operando de 16 ó 32 bits, y el prefijo de operando conmuta entre operandos de 32 o de 16 bits.

La propia instrucción incluye los siguiente campos:

- **Codop:** codop de uno o dos bytes. El codop puede incluir también bits que indican si el operando es de un byte o de tamaño completo (16 ó 32 bits dependiendo del contexto), la dirección de la operación con los datos (a o desde memoria), y si el campo de dato inmediato debe o no extenderse con el signo.
- **ModR/m:** este byte, y el siguiente, aportan información de direccionamiento. El byte «ModR/m» indica si un operando está en un registro o en memoria; si está en memoria, los campos del byte especifican el modo de direccionamiento a utilizar. El byte «ModR/m» consta de tres campos: el campo Mod (dos bits) se combina con el campo r/m para formar 32 valores posibles: 8 a registro y 24 modos de indexado; el campo Reg/Codop (tres bits) especifica o bien un número de registro o tres bits más de información del codop; el campo r/m (3 bits) puede especificar un registro como posición de un operando, o puede formar parte de la codificación del modo de direccionamiento, en combinación con el campo Mod.
- **SIB:** cierta codificación del byte ModR/m indica la inclusión del byte SIB para especificar por completo el modo de direccionamiento. El byte SIB consta de tres campos: El campo de escala (2 bits) especifica el factor de escala para el indexado escalado; el campo de índice (3 bits) especifica el registro índice; el campo de base (3 bits) especifica el registro base.
- **Desplazamiento:** cuando el indicador de modo de direccionamiento especifica que se emplea desplazamiento, se suma un campo de desplazamiento entero con signo de 8, 16 o 32 bits.
- **Inmediato:** proporciona el valor de un operando de 8, 16 o 32 bits.

Algunas comparaciones pueden ser útiles aquí. En el formato del Pentium, el modo de direccionamiento se da como parte de la secuencia de codop, en lugar de con cada operando. Ya que solo un operando puede tener información de modo de dirección, en una instrucción solo puede referenciarse un operando de memoria. En contraste, el VAX lleva la información de modo de dirección con cada operando, permitiendo operaciones de memoria a memoria. Las instrucciones del Pentium son por tanto más compactas. Sin embargo, si se necesita una operación de memoria a memoria, el VAX puede efectuarla con una sola instrucción.

El formato del Pentium permite el uso de desplazamientos para indexado no solo de un byte, sino de dos y de cuatro bytes. Aunque el utilizar desplazamientos de índice mayores tiene como resultado instrucciones más largas, esta característica proporciona la flexibilidad necesaria. Por ejemplo, es útil para direccionar matrices largas o marcos de pila grandes. En contraste, el formato de instrucción del IBM S/370 permite desplazamientos no mayores de 4 KB (doce bits de información para desplazamiento), y el desplazamiento debe ser positivo. Cuando una posición no se puede alcanzar con este desplazamiento, el compilador debe producir código extra para generar la dirección necesaria. Este problema se manifiesta especialmente al trabajar con marcos de pila que tengan más de 4 KB de variables locales. Como se afirma en [DEWA90]: «Como consecuencia de la restricción, generar código para el 370 es tan penoso que ha habido incluso compiladores para el 370 que simplemente eligieron limitar el tamaño del marco de pila a 4 KB».

Como puede verse, la codificación del conjunto de instrucciones del Pentium es muy compleja. Esto se debe en parte a la necesidad de ser compatible con sus predecesores 8086, y en parte al deseo de algunos diseñadores de suministrar todas las ayudas posibles al diseñador del compilador para que produzca código eficiente. Es un tema de controversia si un repertorio de instrucciones tan complejo como éste es preferible al repertorio de un RISC.

FORMATOS DE INSTRUCCIÓN DEL POWERPC

Todas las instrucciones del PowerPC tienen una longitud de 32 bits y siguen un formato regular. Los primeros seis bits especifican la operación a efectuar. En algunos casos, hay una ampliación al codop en alguna parte de la instrucción, que especifica un caso particular de la operación. En la Figura 11.9, los bits de codop se representan mediante la parte sombreada de cada formato.

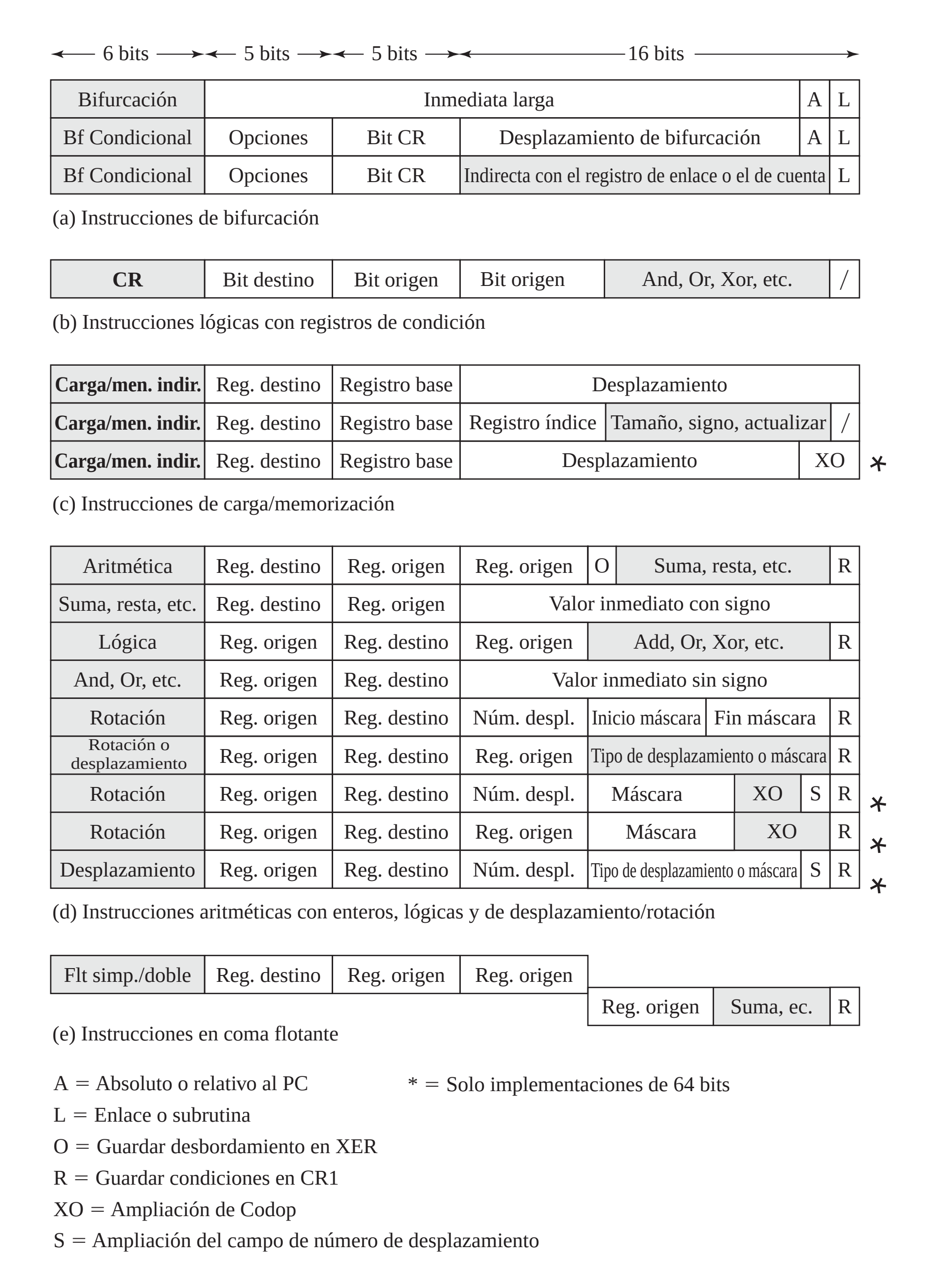


Figura 11.9. Formatos de instrucción del PowerPC.

Observe la estructura regular de los formatos, que facilita el trabajo de las unidades de ejecución de instrucciones. Para todas las instrucciones de carga/memorización, aritméticas, y lógicas, el codop viene seguido de dos referencias a registros de cinco bits cada una, posibilitando el uso de 32 registros de uso general.

Las instrucciones de bifurcación incluyen un bit de enlace (L) que indica que la dirección efectiva de la instrucción siguiente a la de bifurcación va a ubicarse en el registro de enlace. Dos formas de la instrucción incluyen también un bit (A) que indica si el modo de direccionamiento es absoluto o relativo al PC. Para las instrucciones de salto condicional, el campo de bits CR especifica el bit del registro de condición a comprobar. El campo de opciones especifica las condiciones bajo las que va a producirse la bifurcación. Pueden especificarse las siguientes condiciones:

- Saltar siempre.
- Saltar si cuenta $\neq 0$ y la condición es falsa.
- Saltar si cuenta $\neq 0$ y la condición es verdadera.
- Saltar si cuenta $= 0$ y la condición es falsa.
- Saltar si cuenta $= 0$ y la condición es verdadera.
- Saltar si cuenta $\neq 0$.
- Saltar si cuenta $= 0$.
- Saltar si la condición es falsa.
- Saltar si la condición es verdadera.

La mayoría de las instrucciones que efectúan cálculos (aritméticas, aritméticas en coma flotante, lógicas) incluyen un bit que indica si el resultado de la operación debiera grabarse en el registro de condición. Como se mostrará, esta posibilidad es útil para procesar la predicción de saltos.

Las instrucciones en coma flotante tienen campos para tres registros origen. En muchos casos solo se emplean dos registros origen. Unas cuantas instrucciones implican multiplicar dos registros origen, y después sumar o restar de otro registro origen. Se han incluido estas instrucciones compuestas por ser de uso bastante frecuente. Por ejemplo, con instrucciones de multiplicación-suma puede implementarse el producto escalar que forma parte de muchas operaciones con matrices.

11.5. LECTURAS RECOMENDADAS

Las referencias citadas en el Capítulo 10 son igualmente aplicables para el material de este capítulo. En [BLAA97] se incluye una descripción detallada de formatos de instrucciones y de modos de direccionamiento. Además, puede que al lector le interese consultar [FLYN85] para una discusión y análisis sobre aspectos de diseño de repertorios de instrucciones, particularmente aquellos relativos a los formatos.

BLAA97 Blaauw, G. y Brooks, F.: *Computer Architecture: Concepts and Evolution.* Reading, MA. Addison-Wesley, 1997.

FLYN85 Flynn, M.; Johnson, J. y Wakwfield, S.: «On Instruction Sets and Their Formats». *IEEE Transactions on Computers,* marzo, 1985.

PALABRAS CLAVE, PREGUNTAS DE REPASO Y PROBLEMAS

PALABRAS CLAVE

autoindexado dirección con desplazamiento direccionamiento con registro base direccionamiento de registros direccionamiento directo	dirección efectivadireccionamiento inmediato direccionamiento inmediato direccionamiento indirecto con registro	direccionamiento relativo formato de instrucción indexado palabra postindexado preindexado

PREGUNTAS DE REPASO

11.1. Defina brevemente direccionamiento inmediato.
11.2. Defina brevemente direccionamiento directo.
11.3. Defina brevemente direccionamiento indirecto.
11.4. Defina brevemente direccionamiento de registros.
11.5. Defina brevemente direccionamiento indirecto con registro.
11.6. Defina brevemente direccionamiento con desplazamiento.
11.7. Defina brevemente direccionamiento relativo.
11.8. ¿Qué ventaja tiene el autoindexado?
11.9. ¿Qué diferencia existe entre preindexado y postindexado?
11.10. ¿Qué hechos influyen al determinar el uso de los bits de direcciones de una instrucción?
11.11. ¿Cuáles son las ventajas y desventajas del uso de un formato de instrucción de longitud variable?

PROBLEMAS

11.1. Dados los valores de memoria siguientes, y suponiendo una máquina con instrucciones de una sola dirección, y con un acumulador, ¿qué valores cargan las siguientes instrucciones en el acumulador?

- La palabra 20 contiene 40.
- La palabra 30 contiene 50.
- La palabra 40 contiene 60.
- La palabra 50 contiene 70.

(a) CARGA INMEDIATA 20
(b) CARGA DIRECTA 20
(c) CARGA INDIRECTA 20
(d) CARGA INMEDIATA 30
(e) CARGA DIRECTA 30
(f) CARGA INDIRECTA 30

11.2. Suponga que la dirección almacenada en el contador de programa se designa con el símbolo X1. La instrucción almacenada en X1 tiene una parte de dirección (referencia a operando) X2. El operando que se necesita para ejecutar la instrucción está almacenado en la palabra de memoria con dirección X3. Un registro de índice contiene el valor X4. ¿Qué relación existe entre estas cantidades si el modo de direccionamiento de la instrucción es (a) directo; (b) indirecto; (c) relativo al PC; (d) indexado?

11.3. El campo de dirección de una instrucción contiene el valor decimal 14. Indique dónde se ubica el operando correspondiente en cada uno de los casos:

(a) direccionamiento inmediato
(b) direccionamiento directo
(c) direccionamiento indirecto
(d) direccionamiento de registro
(e) direccionamiento indirecto con registro

11.4. Considere un procesador de 16 bits para el que se tiene en memoria principal, a partir de la dirección 200, lo siguiente:

200	Cargar en AC	Modo
201	500	
202	Instrucción siguiente	

La primera parte de la primera palabra indica que es una instrucción que carga un valor en el acumulador. El campo «Modo» indica un modo de direccionamiento y, si procede, indica un registro origen; suponga que cuando se usa, el registro origen es R1, que tiene un valor de 400. Hay también un registro base que contiene el valor 100. El valor 500 ubicado en la posición 201 puede formar parte del cálculo de la dirección. Suponga que la posición 399 contiene el valor 999, la posición 400 contiene el valor 1000 y así sucesivamente. Determine la dirección efectiva, y el operando que se carga, para cada uno de los siguientes modos de direccionamiento:

a. Directo **b.** Inmediato **c.** Indirecto. Relativo al PC **e.** Con desplazamiento	**f.** Registro. Indirecto con registro **h.** Autoindexado con incremento, empleando R1

11.5. Una instrucción de bifurcación con modo relativo al PC tiene una longitud de tres bytes. La dirección de la instrucción es 256028, en decimal. Determine la posición a la que se salta cuando el desplazamiento, con signo, de la instrucción es -31.

11.6. Una instrucción de bifurcación con modo relativo al PC está almacenada en la posición de memoria 530_{10}. El salto se efectúa a la posición 530_{10}. El campo de dirección de la instrucción es de 10 bits. ¿Cuál es el valor binario de la instrucción?

11.7. ¿Cuántas referencias a memoria necesita efectuar el procesador cuando capta y ejecuta una instrucción con modo de direccionamiento indirecto, si dicha instrucción es (a) un cálculo que requiere de un solo operando; (b) un salto?

11.8. El IBM 370 no permite direccionamiento indirecto. Suponga que la dirección de un operando está en memoria principal. ¿Cómo podría accederse al operando?

11.9. En [COOK82], el autor propone que los modos relativos al PC se eliminen en favor de otros modos, tales como el uso de una pila ¿Cuál es la desventaja de esta propuesta?

11.10. El Pentium incluye la siguiente instrucción:

IMUL op1, op2, immediate

Esta instrucción multiplica op2, que puede ser o registro o memoria, por un valor de operando inmediato, y guarda el resultado en op1, que debe ser un registro. No hay otra instrucción con tres operandos de este tipo en el repertorio. ¿Qué posible uso tiene esta instrucción? (*Nota:* considere el indexado).

11.11. Considere un procesador que incluye un modo de direccionamiento base con indexado. Suponga que se encuentra una instrucción que emplea este modo de direccionamiento, y que especifica un desplazamiento de 1970, en decimal. En el momento actual el registro de base y de índice contienen respectivamente los número decimales 48022 y 8. ¿Cuál es la dirección del operando?

11.12. EA = (X)+ define como dirección efectiva el contenido de la posición X, haciendo que se incremente X en un valor igual a la longitud de palabra tras calcular la dirección efectiva. EA = −(X) define como dirección efectiva el contenido de la posición X, haciendo que se decremente X en un valor igual a la longitud de palabra antes de calcular la dirección efectiva; EA = (X)− define como dirección efectiva el contenido de la posición X, haciendo que se decremente X en un valor igual a la longitud de palabra tras calcular la dirección efectiva. Considere las siguientes instrucciones, cada una de ellas con el formato (Operación operando_origen, operando_destino), y el resultado de la operación ubicado en el operando destino:

(a) OP X, (X)
(b) OP X, (X) +
(c) OP (X) +, (X)
(d) OP − (X), (X)
(e) OP − (X), (X) +
(f) OP (X) +, (X) +
(g) OP (X) −, (X)

Usando X como puntero de pila, ¿cuál de las instrucciones anteriores puede desapilar los dos elementos de la cabecera de la pila, realizar el cálculo indicado (por ejemplo, sumar operandos origen y destino, y memorizar el resultado en destino) y apilar de nuevo el resultado en la pila? Para cada una de las instrucciones, ¿la pila crece hacia la dirección 0 de memoria o en sentido contrario?

11.13. Suponga un procesador orientado al uso de pila que incluye las operaciones con la pila: PUSH y POP. Las operaciones aritméticas implican automáticamente al elemento, o la pareja de elementos, de la cabecera de la pila. Partiendo de una pila vacía, ¿qué elementos quedan en la pila después de ejecutarse las siguientes instrucciones?

```
PUSH   4
PUSH   7
PUSH   8
ADD
PUSH   10
SUB
MUL
```

11.14. Justifique la afirmación de que una instrucción de 32 bits es probablemente mucho menos del doble de útil que una de 16 bits.

11.15. ¿Por qué fue drástica la decisión de IBM de pasar de 36 a 32 bits por palabra? ¿Para quién lo fue?

11.16. Suponga un repertorio de instrucciones que utiliza una longitud de instrucción de 16 bits. Los operandos se especifican con seis bits. Hay K instrucciones de dos operandos y L instrucciones de cero operandos. ¿Cuál es el número máximo de instrucciones de un operando que pueden permitirse?

11.17. Diseñe un código de operación de longitud variable que permita codificar todas las instrucciones siguientes con una longitud de instrucción de 36 bits:

- Instrucciones con dos direcciones de 15 bits y un número de registro de 3 bits.
- Instrucciones con una dirección de 15 bits y un número de registro de 3 bits.
- Instrucciones sin direcciones ni registros.

11.18. Considere los resultados del Problema 10.6. Suponga que M es una dirección de memoria de 16 bits y que X, Y, y Z son o bien direcciones de 16 bits o números de registros de 4 bits. La máquina de una dirección emplea un acumulador, y las máquinas de dos y tres direcciones tienen 16 registros e instrucciones que operan con todas las combinaciones de posiciones de memoria y de registros. Suponiendo codops de ocho bits y longitudes de instrucción múltiplos de cuatro bits, ¿cuántos bits necesita cada máquina para calcular X?

11.19. ¿Hay alguna justificación posible para una instrucción con dos codops?

11.20. El Zilog Z8001, de 16 bits, tiene el siguiente formato general de instrucción:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Modo		Codop					w/b	Operando 2				Operando 1			

El campo *mode* especifica cómo localizar los operandos de los campos *operando*. El campo *w/b* (word/byte) se usa en ciertas instrucciones para indicar si los operandos son bytes o palabras de 16 bits. El campo *operando 1* puede (dependiendo del contenido del campo *modo*) especificar uno de los 16 registros de uso general. El campo *operando 2* puede especificar cualquier registro de uso general excepto el registro 0. Cuando el campo *operando 2* se completa con ceros, cada uno de los codops originales adquiere un significado nuevo.

(a) ¿De cuantos codops dispone el Z8001?

(b) Sugiera una forma eficiente de disponer de más codops e indique los compromisos a adoptar.

CAPÍTULO 12

Estructura y funcionamiento del procesador

PUNTOS CLAVE

- ➤ Un procesador incluye tanto registros visibles por el usuario como registros de control/estado. Los primeros pueden referenciarse, implícita o explícitamente, en las instrucciones máquina. Los registros visibles por el usuario pueden ser de uso general o tener una utilidad especial, tal como almacenamiento de números en coma fija o coma flotante, direcciones, índices o punteros de segmento. Los registros de control y de estado se usan para controlar el funcionamiento del procesador. Un claro ejemplo es el contador de programa. Otro ejemplo importante es la palabra de estado del programa (*program status word*, PSW) que contiene diversos bits de estado y condición. Estos incluyen bits para reflejar el resultado de la operación aritmética más reciente, bits de habilitación de interrupciones y un indicador de cuándo el procesador funciona en modo supervisor o usuario.
- ➤ Los procesadores utilizan la segmentación de instrucciones para acelerar la ejecución. Fundamentalmente, la segmentación de cauce supone dividir el ciclo de instrucción en varias etapas separadas que operan secuencialmente, tales como captación de instrucción, decodificación de instrucción, cálculo de direcciones de operandos, captación de operandos, ejecución de instrucción y escritura del operando resultado. Las instrucciones pasan a través de estas etapas como en una cadena de montaje, de modo que en principio cada etapa puede estar trabajando en una instrucción diferente al mismo tiempo. La existencia de saltos y dependencias entre instrucciones complica el diseño y el uso de los cauces segmentados.

Este capítulo trata aspectos del procesador no cubiertos en la Parte Tres y establece el escenario para la discusión de los RISC y de la arquitectura superescalar en los Capítulos 13 y 14.

Comenzamos con un resumen de la organización del procesador. Después analizaremos los registros, que constituyen la memoria interna del procesador. Estaremos entonces en condiciones de volver a la discusión (comenzada en la Sección 3.2) sobre el ciclo de instrucción. Completan nuestro estudio una descripción del ciclo de instrucción y una técnica usual conocida como segmentación (*pipelining*) de instrucciones. El capítulo concluye con un examen de algunos aspectos adicionales de las organizaciones del Pentium y del PowerPC.

12.1. ORGANIZACIÓN DEL PROCESADOR

Para comprender la organización del procesador, consideremos los requisitos que ha de cumplir:

- **Captar instrucción:** el procesador lee una instrucción de la memoria (registro, caché o memoria principal).
- **Interpretar instrucción:** la instrucción se decodifica para determinar qué acción es necesaria.

- **Captar datos:** la ejecución de una instrucción puede exigir leer datos de la memoria o de un módulo de E/S.
- **Procesar datos:** la ejecución de una instrucción puede exigir llevar a cabo alguna operación aritmética o lógica con los datos.
- **Escribir datos:** los resultados de una ejecución pueden exigir escribir datos en la memoria o en un módulo de E/S.

Para hacer estas cosas, es obvio que el procesador necesita almacenar algunos datos temporalmente. Debe recordar la posición de la última instrucción de forma que pueda saber de dónde tomar la siguiente. Necesita almacenar instrucciones y datos temporalmente mientras una instrucción está ejecutándose. En otras palabras, el procesador necesita una pequeña memoria interna.

La Figura 12.1 es una visión simplificada de un procesador, que indica su conexión con el resto del sistema a través del bus del sistema. Sería necesaria una interfaz similar para cualquiera de las estructuras de interconexión descritas en el Capítulo 3. El lector recordará que los principales componentes del procesador son una *unidad aritmético lógica* (*arithmetic and logic unit*, ALU) y una *unidad de control* (*control unit*, CU). La ALU lleva a cabo el verdadero cálculo o procesamiento de datos. La unidad de control controla las transferencias de datos e instrucciones hacia dentro y hacia fuera del procesador, y el funcionamiento de la ALU. Además, la figura muestra una memoria interna mínima, que consta de un conjunto de posiciones de almacenamiento llamadas *registros*.

La Figura 12.2 presenta una visión un poco más detallada del procesador. Se indican los caminos de transferencia de datos y de la lógica de control, que incluyen un elemento con el rótulo *bus interno del procesador*. Este elemento es necesario para transferir datos entre los diversos registros y la ALU, ya que la ALU en realidad solo opera con datos de la memoria interna del procesador. La figura muestra también los elementos básicos típicos de la ALU. Observe la similitud entre la estructura interna del computador en su totalidad y la estructura interna del procesador. En ambos casos hay una pequeña colección de elementos principales (computador: procesador, E/S, memoria; procesador: unidad de control, ALU, registros) conectados por caminos de datos.

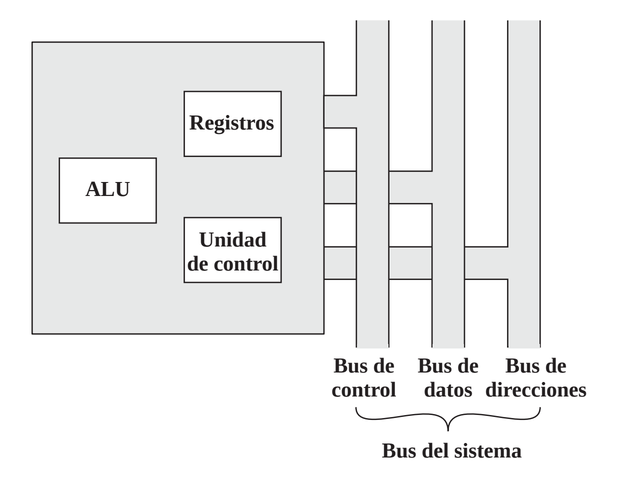


Figura 12.1. El procesador y el bus del sistema.

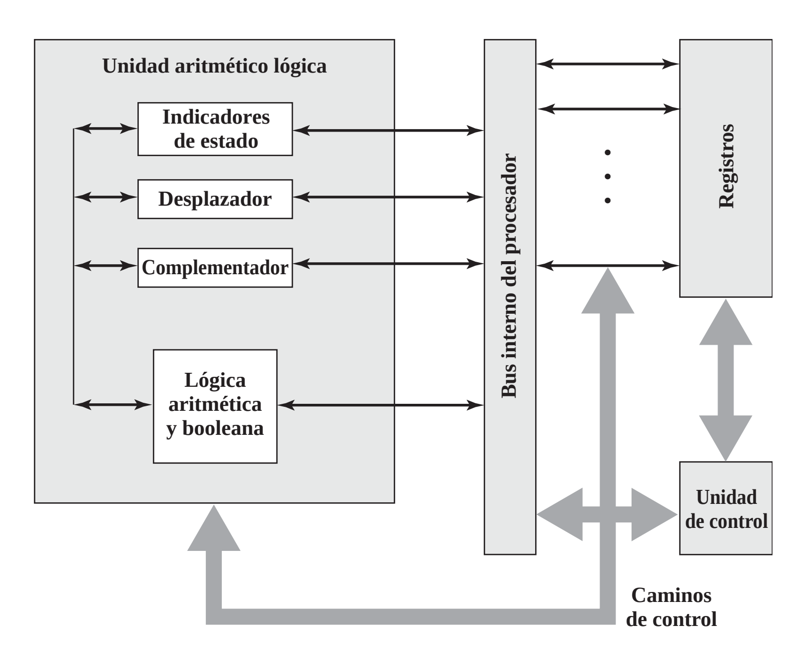


Figura 12.2. Estructura interna del procesador.

12.2. ORGANIZACIÓN DE LOS REGISTROS

Como discutimos en el Capítulo 4, un computador emplea una jerarquía de memoria. En los niveles más altos de la jerarquía, la memoria es más rápida, más pequeña y más cara (por bit). Dentro del procesador hay un conjunto de registros que funciona como un nivel de memoria por encima de la memoria principal y de la caché en la jerarquía. Los registros del procesador son de dos tipos:

- **Registros visibles por el usuario:** permiten al programador de lenguaje máquina o de ensamblador minimizar las referencias a memoria principal por medio de la optimización del uso de registros.
- **Registros de control y de estado:** son utilizados por la unidad de control para controlar el funcionamiento del procesador y por programas privilegiados del sistema operativo para controlar la ejecución de programas.

No hay una separación bien definida de registros dentro de estas dos categorías. Por ejemplo, en algunas máquinas el contador de programa es visible por el usuario (por ejemplo, en el Pentium), pero en muchas no lo es. Para el objetivo de la siguiente discusión, no obstante, usaremos estas categorías.

REGISTROS VISIBLES POR EL USUARIO

Un registro visible por el usuario es aquél que puede ser referenciado por medio del lenguaje máquina que ejecuta el procesador. Podemos clasificarlos en las siguientes categorías:

- Uso general
- Datos
- Direcciones
- Códigos de condición

Los **registros de uso general** pueden ser asignados por el programador a diversas funciones. A veces, su uso dentro del repertorio de instrucciones es ortogonal a la operación. Es decir, cualquier registro de uso general puede contener el operando para cualquier código de operación. Esto proporciona una utilización de registros de uso general auténtico. Con frecuencia, sin embargo, existen restricciones. Por ejemplo, puede haber registros específicos para operaciones en coma flotante y para operaciones con la pila.

En algunos casos los registros de uso general pueden utilizarse para funciones de direccionamiento (por ejemplo, en direccionamientos indirectos por medio de registro o con desplazamiento). En otros casos hay una separación parcial o total entre registros de datos y registros de direcciones. Los **registros de datos** pueden usarse únicamente para contener datos y no se pueden emplear en el cálculo de la dirección de un operando. Los **registros de dirección** pueden ser de uso más o menos general, o pueden estar dedicados a un modo de direccionamiento particular. Entre otros, se pueden citar los siguientes ejemplos:

- **Punteros de segmento:** en una máquina con direccionamiento segmentado (*véase* Sección 8.3) un registro de segmento contiene la dirección de la base del segmento. Puede haber múltiples registros: por ejemplo uno para el sistema operativo y otro para el proceso actual.
- **Registros índice:** se usan para direccionamiento indexado y pueden ser autoindexados.
- **Puntero de pila:** si existe direccionamiento a pila visible por el usuario, normalmente hay un registro dedicado que apunta a la cabecera de ésta. Ello permite un direccionamiento implícito; es decir, apilar (*push*), desapilar (*pop*) y otras instrucciones de la pila no necesitan contener un operando explícito referente a ella.

Hay varias cuestiones de diseño a tratar en este punto. Un importante asunto es decidir si utilizar registros de uso completamente general o si especializar su uso. Nos referimos ya a este asunto en el capítulo anterior, dado que afecta al diseño del repertorio de instrucciones. Con el uso de registros especializados, generalmente puede quedar implícito en el código de operación a qué tipo de registro se refiere un determinado campo de operando. El campo de operando solo tiene que identificar un registro de un conjunto de registros especializados en lugar de uno de entre todos los registros, lo cual ahorra bits. Por otra parte, esta especialización limita la flexibilidad del programador.

Otro problema de diseño es el número de registros, de uso general o de datos más direcciones, que tienen que incluirse. De nuevo, ello afecta al diseño del repertorio de instrucciones, ya que si hay más registros se necesitan más bits para el campo de operando. Como discutimos anteriormente, parece óptimo disponer de un número de registros entre 8 y 32 [LUND77]. Menos registros se traducen en más referencias a memoria; más registros no reducen significativamente las referencias a memoria (*véase*, por ejemplo [WILL90]). No obstante, existe una nueva aproximación, que saca partido al uso de cientos de registros, utilizada en algunos sistemas RISC y estudiada en el Capítulo 13.

Por último, está la cuestión del tamaño de los registros. Los registros que han de contener direcciones deben ser lo suficientemente grandes como para albergar la dirección mayor. Los registros de

datos deben ser capaces de contener valores de la mayoría de tipos de datos. Algunas máquinas permiten que dos registros contiguos sean usados como uno solo para contener valores de tamaño doble.

Una última categoría de registros, que es al menos parcialmente visible por el usuario, contiene **códigos de condición** (también llamados *indicadores* o *«flags»*). Los códigos de condición son bits fijados por el hardware del procesador como resultado de alguna operación. Por ejemplo, una operación aritmética puede producir un resultado positivo, negativo, nulo o con desbordamiento. Además de almacenarse el propio resultado en un registro o en la memoria, se obtiene también un código de condición. El código puede ser examinado con posterioridad como parte de una operación de salto condicional.

Los bits de códigos de condición se agrupan en uno o más registros. Normalmente forman parte de un registro de control. Por lo general, las instrucciones máquina permiten que estos bits sean leídos por referencia implícita, pero el programador no puede alterarlos.

Muchos procesadores, incluyendo los basados en la arquitectura IA-64 y los procesadores MIPS, no usan códigos de condición en absoluto. En lugar de ello, las instrucciones de salto condicional especifican una comparación que debe hacerse y actúan de acuerdo con el resultado de la comparación, sin almacenar ningún código de condición. La Tabla 12.1, basada en [DERO87], enumera las principales ventajas e inconvenientes de los códigos de condición.

Tabla 12.1. Códigos de condición.

Ventajas	Inconvenientes
1. Dado que los códigos de condición son fijados por instrucciones aritméticas y de movimiento de datos normales, deberían reducir el número de instrucciones COMPARE y TEST necesarias. **2.** Las instrucciones condicionales tales como BRANCH se simplifican comparadas con las instrucciones compuestas tales como TEST AND BRANCH. **3.** Los códigos de condición facilitan los saltos múltiples. Por ejemplo, una instrucción TEST puede venir seguida de dos saltos, uno si menor o igual que cero y otro si mayor que cero.	**1.** Los códigos de condición añaden complejidad tanto al hardware como al software. Los bits de códigos de condición se modifican con frecuencia de distintas maneras por instrucciones distintas, complicando la vida tanto al microprogramador como al escritor de compiladores. **2.** Los códigos de condición son irregulares; normalmente no forman parte del camino de datos principal, necesitando conexiones hardware adicionales. **3.** Frecuentemente, las máquinas con códigos de condición tienen que añadir de todos modos instrucciones especiales «sin códigos de condición» para situaciones especiales, tales como comprobación de bits, control de bucles y operaciones atómicas con semáforos. **4.** En una implementación segmentada, los códigos de condición requieren una sincronización especial para evitar conflictos.

En algunas máquinas, una llamada a una subrutina hará que se guarden automáticamente todos los registros visibles por el usuario, que serán restablecidos en el retorno de la subrutina. El procesador lleva a cabo las operaciones de guardar y restablecer los registros como parte de la ejecución de las instrucciones de llamada y retorno. Ello permite a cada subrutina usar los registros visibles por el usuario de manera independiente. En otras máquinas, es responsabilidad del programador guardar los contenidos de los registros visibles por el programador que sean relevantes antes de la llamada a la subrutina, incluyendo en el programa instrucciones para este fin.

REGISTROS DE CONTROL Y DE ESTADO

Hay diversos registros del procesador que se emplean para controlar su funcionamiento. La mayoría de ellos, en la mayor parte de las máquinas, no son visibles por el usuario. Algunos de ellos pueden ser visibles por ciertas instrucciones máquina ejecutadas en un modo de control o de sistema operativo.

Naturalmente, máquinas diferentes tendrán distintas organizaciones de registros y usarán distinta terminología. A continuación se presenta una lista razonablemente completa de tipos de registros, con una breve descripción.

Son esenciales cuatro registros para la ejecución de una instrucción:

- **Contador de programa (*Program Counter,* PC):** contiene la dirección de la instrucción a captar.
- **Registro de instrucción (*Instruction Register,* IR):** contiene la instrucción captada más recientemente.
- **Registro de dirección de memoria (*Memory Address Register,* MAR):** contiene la dirección de una posición de memoria.
- **Registro intermedio de memoria (*Memory Buffer Register,* MBR):** contiene la palabra de datos a escribir en memoria o la palabra leída más recientemente.

No todos los procesadores tienen registros internos designados como MAR y MBR, pero es necesario algún mecanismo de almacenamiento intermedio equivalente mediante el cual se dé salida a los bits que van a ser transferidos al bus del sistema y se almacenen temporalmente los bits leídos del bus de datos.

Típicamente, el procesador actualiza PC después de cada captación de instrucción de manera que siempre apunta a la siguiente instrucción a ejecutar. Una instrucción de bifurcación o salto también modificará el contenido de PC. La instrucción captada se carga en IR, donde son analizados el código de operación y los campos de operando. Se intercambian datos con la memoria por medio de MAR y de MBR. En un sistema con organización de bus, MAR se conecta directamente al bus de direcciones, y MBR directamente al bus de datos. Los registros visibles por el usuario repetidamente intercambian datos con MBR.

Los cuatro registros que se acaban de mencionar se usan para la transferencia de datos entre el procesador y la memoria. Dentro del procesador, los datos tienen que ofrecerse a la ALU para su procesamiento. La ALU puede tener acceso directo a MBR y a los registros visibles por el usuario. Como alternativa, puede haber registros intermedios adicionales en torno a la ALU; estos registros sirven como registros de entrada y salida de la ALU e intercambian datos con MBR y los registros visibles por el usuario.

Muchos diseños de procesadores incluyen un registro o un conjunto de registros, conocidos a menudo como *palabra de estado del programa* (PSW, *program status word*), que contiene información de estado. PSW contiene típicamente códigos de condición además de otra información de estado. Entre los campos o indicadores comunes se incluyen los siguientes:

- **Signo:** contiene el bit de signo del resultado de la última operación aritmética.
- **Cero**: puesto a uno cuando el resultado es 0.
- **Acarreo:** puesto a uno si una operación da lugar a un acarreo (en la suma) o adeudo (en la resta) del bit más significativo. Se usa en operaciones aritméticas multipalabra.
- **Igual:** puesto a uno si el resultado de una comparación lógica es la igualdad.
- **Desbordamiento:** usado para indicar un desbordamiento aritmético.
- **Interrupciones habilitadas/inhabilitadas:** usado para permitir o inhabilitar interrupciones.
- **Supervisor:** indica si el procesador funciona en modos supervisor o usuario. Únicamente en modo supervisor se pueden ejecutar ciertas instrucciones privilegiadas y se puede acceder a ciertas áreas de memoria.

En algún diseño concreto de procesador es posible encontrar otros registros relativos a estado y control. Puede existir un puntero a un bloque de memoria que contenga información de estado adicional (por ejemplo, bloques de control de procesos). En las máquinas que usan interrupciones vectorizadas puede existir un registro de vector de interrupción. Si se utiliza una pila para llevar a cabo ciertas funciones (por ejemplo, llamada a subrutina), se necesita un puntero de pila del sistema. En un sistema de memoria virtual se usa un puntero a la tabla de páginas. Por último, pueden emplearse registros para el control de operaciones de E/S.

En el diseño de la organización de los registros de control y estado entran en juego varios factores. Una cuestión primordial es el soporte del sistema operativo. Algunos tipos de información de control son de utilidad específica para el sistema operativo. Si el diseñador del procesador posee una comprensión funcional del sistema operativo que se va a utilizar, la organización de los registros puede adaptarse hasta cierto punto a ese sistema operativo.

Otra decisión importante en el diseño es la distribución de información de control entre registros y memoria. Es frecuente dedicar los primeros (más bajos) pocos cientos o miles de palabras de memoria para fines de control. El diseñador debe decidir cuánta información de control debiera estar en registros y cuánta en memoria. Se presenta el compromiso habitual entre coste y velocidad.

EJEMPLOS DE ORGANIZACIONES DE REGISTROS DE MICROPROCESADORES

Resulta instructivo examinar y comparar las organizaciones de registros de sistemas análogos. En esta sección, examinamos dos microprocesadores de 16 bits que fueron diseñados aproximadamente al mismo tiempo: el Motorola MC68000 [STRI79] y el Intel 8086 [MORS78]. Las Figuras 12.3a y b representan la organización de registros de cada uno de ellos; los registros estrictamente internos, tales como el registro de dirección de memoria, no se muestran.

El MC68000 distribuye sus registros de 32 bits en ocho de datos y nueve de direcciones. Los ocho registros de datos se usan principalmente para manipulación de datos y también se usan en

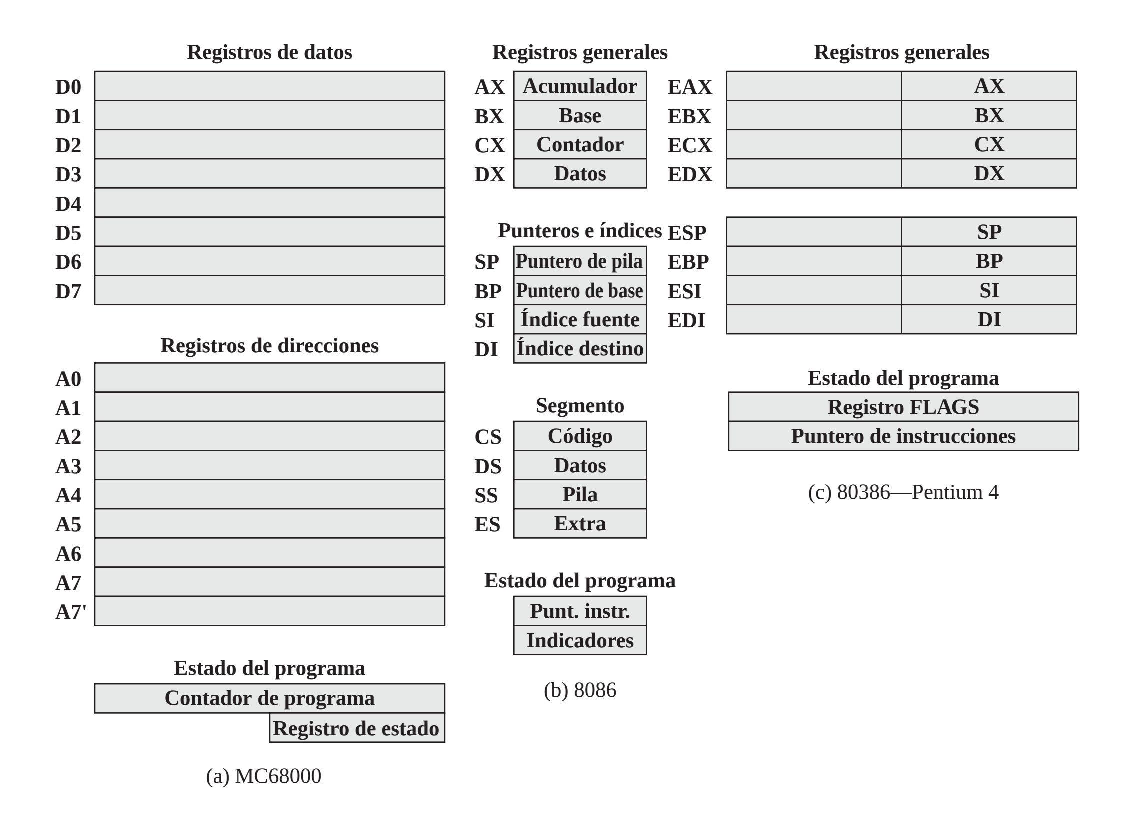


Figura 12.3. Ejemplos de organizaciones de registros de microprocesadores.

direccionamiento como registros índice. El ancho de los registros permite operaciones con datos de 8, 16 y 32 bits, según determine el código de operación. Los registros de direcciones contienen direcciones de 32 bits (no hay segmentación); dos de estos registros se usan también como punteros de pila, uno para los usuarios y el otro para el sistema operativo, dependiendo del modo de ejecución en curso. Los dos registros se referencian como 7, dado que solo uno de ellos puede usarse en un instante dado. El MC68000 también incluye un contador de programa de 32 bits y un registro de estado de 16 bits.

El equipo de Motorola quiso un repertorio de instrucciones muy regular, sin registros de uso especial. Su interés por la eficiencia del código los condujo a dividir los registros en dos componentes funcionales, ahorrando un bit en cada campo de especificación de registro. Parece un compromiso razonable entre generalidad total y compacidad del código.

El Intel 8086 usa un enfoque diferente para la organización de los registros. Cada uno de los registros tiene un uso especial, aunque algunos registros se pueden emplear también para un uso general. El 8086 contiene cuatro registros de datos de 16 bits que son direccionables como registros de bytes o como registros de 16 bits, y cuatro registros punteros e índices de 16 bits. Los registros de datos pueden utilizarse como de uso general en algunas instrucciones. En otras, los registros se usan implícitamente. Por ejemplo, una instrucción de multiplicación siempre usa el acumulador. Los cuatro registros punteros se usan también implícitamente en algunas operaciones; cada uno contiene un desplazamiento dentro de un segmento. Hay también cuatro registros de segmento de 16 bits. Tres de los cuatro registros de segmento se usan de una forma dedicada e implícita para apuntar al segmento de la instrucción en curso (útil para instrucciones de salto), a un segmento que contenga datos, y a un

segmento que contenga una pila, respectivamente. Estos usos dedicados e implícitos proporcionan una codificación compacta con el coste de una flexibilidad reducida. El 8086 incluye también un puntero de instrucciones y un conjunto de indicadores de un bit de estado y de control.

Debe quedar claro qué es lo significativo de esta comparación. No hay, por el momento, una filosofía universalmente aceptada sobre la mejor forma de organizar los registros del procesador [TOON81]. Igual que ocurre en el diseño global del repertorio de instrucciones y en algunos otros aspectos del diseño del procesador, se trata más bien de una cuestión de opinión y de gustos.

En la Figura 12.3c se ilustra un segundo aspecto instructivo acerca del diseño de la organización de los registros. Esta figura muestra la organización de los registros visibles por el usuario en el Intel 80386 [ELAY85], un microprocesador de 32 bits diseñado como una ampliación del 8086[1]. El 80386 usa registros de 32 bits. No obstante, para proporcionar compatibilidad ascendente para los programas escritos en la primera máquina, el 80386 conserva la organización de registros original integrada en la nueva organización. Dada esta restricción en el diseño, los arquitectos de los procesadores de 32 bits han limitado la flexibilidad al diseñar la organización de los registros.

12.3. CICLO DE INSTRUCCIÓN

En la Sección 3.2, describimos el ciclo de instrucción del procesador (Figura 3.9). Recordemos que un ciclo de instrucción incluye los siguientes subciclos:

- **Captación:** llevar la siguiente instrucción de la memoria al procesador.
- **Ejecución:** interpretar el código de operación y llevar a cabo la operación indicada.
- **Interrupción:** si las interrupciones están habilitadas y ha ocurrido una interrupción, guardar el estado del proceso actual y atender la interrupción.

Estamos ahora en condiciones de dar alguna explicación más acerca del ciclo de instrucción. En primer lugar, debemos introducir un subciclo adicional, conocido como el ciclo indirecto.

EL CICLO INDIRECTO

Hemos visto, en el Capítulo 11, que la ejecución de una instrucción puede involucrar a uno o más operandos en memoria, cada uno de los cuales requiere un acceso a memoria. Además, si se usa direccionamiento indirecto serán necesarios accesos a memoria adicionales.

Podemos considerar la captación de direcciones indirectas como un subciclo de instrucción más. El resultado se muestra en la Figura 12.4. La principal línea de actividad consiste en alternar las actividades de captación y ejecución de instrucciones. Después de que una instrucción sea captada, es examinada para determinar si incluye algún direccionamiento indirecto. Si es así, los operandos requeridos se captan usando direccionamiento indirecto. Tras la ejecución se puede procesar una interrupción antes de la captación de la siguiente instrucción.

---

[1] Dado que el MC68000 ya usaba registros de 32 bits, el MC68020 [MACG84], que es una ampliación completa a 32 bits, usa la misma organización de registros.

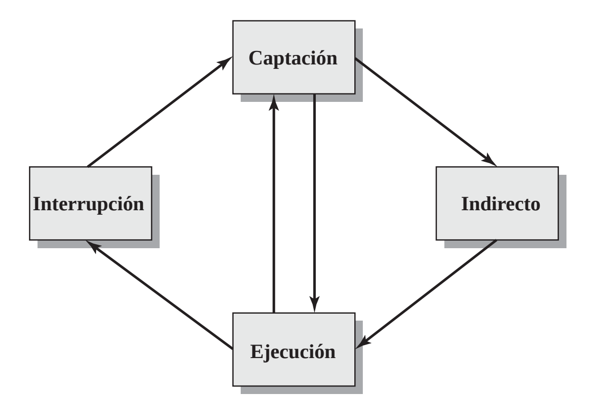


Figura 12.4. El ciclo de instrucción.

En la Figura 12.5, que es una versión revisada de la Figura 3.12, se muestra otra forma de ver este proceso. Esta nueva figura ilustra más correctamente la naturaleza del ciclo de instrucción. Una vez que una instrucción es captada, deben identificarse sus campos de operandos. Se capta entonces de la memoria cada operando de entrada, y este proceso puede requerir direccionamiento indirecto. Los operandos ubicados en registros no necesitan ser captados. Una vez que se ejecuta la operación, puede ser necesario un proceso similar para almacenar el resultado en la memoria principal.

FLUJO DE DATOS

La secuencia exacta de eventos que tienen lugar durante un ciclo de instrucción depende del diseño del procesador. Podemos, no obstante, indicar en términos generales lo que debe ocurrir. Supongamos

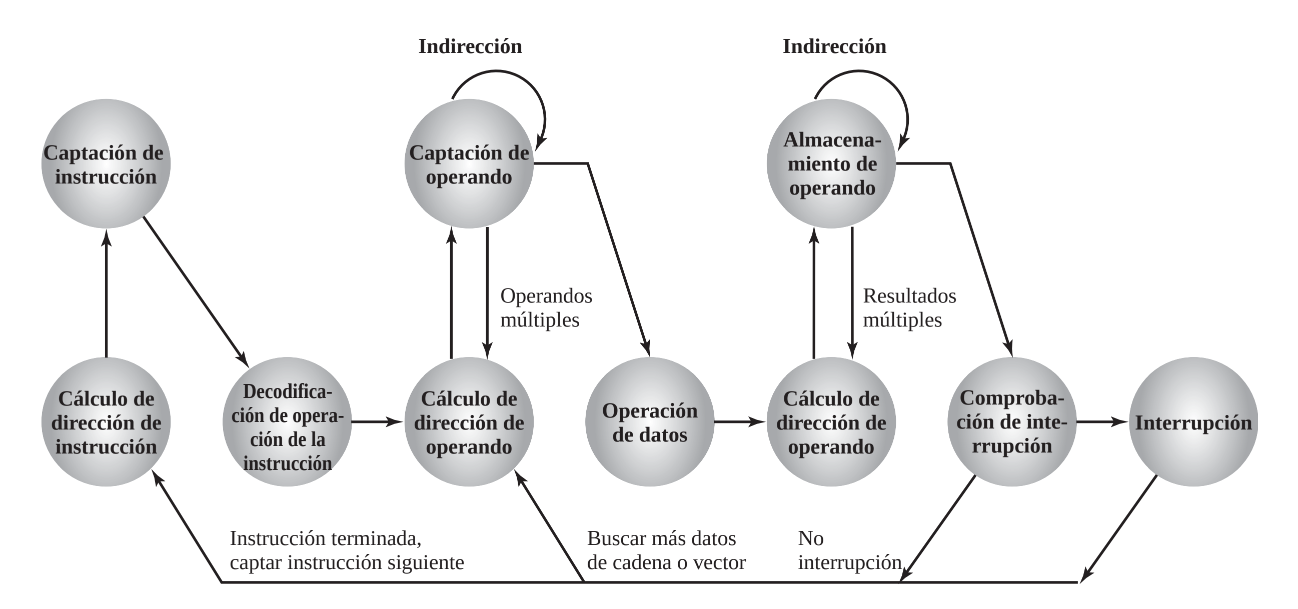


Figura 12.5. Diagrama de estados del ciclo de instrucción.

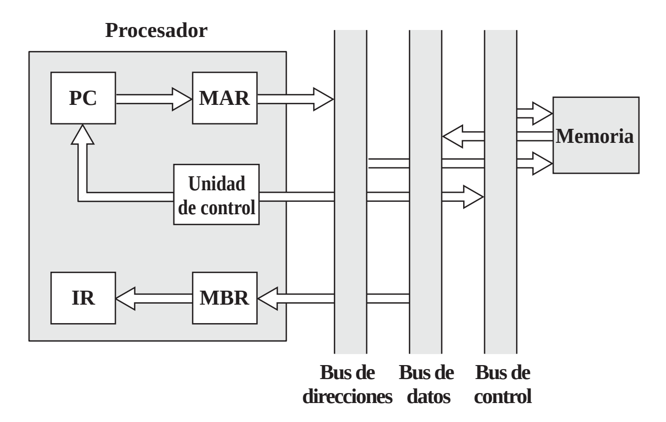


Figura 12.6. Flujo de datos, ciclo de captación.

un procesador que emplee un registro de dirección de memoria (MAR), un registro intermedio de memoria (MBR), un contador de programa (PC) y un registro de instrucción (IR).

Durante el *ciclo de captación* se lee una instrucción de la memoria. La Figura 12.6 muestra el flujo de datos en este ciclo. PC contiene la dirección de la siguiente instrucción que hay que captar. Esta dirección se lleva a MAR y se coloca en el bus de direcciones. La unidad de control solicita una lectura de memoria, y el resultado se pone en el bus de datos, se copia en MBR y después se lleva a IR. Mientras tanto, PC se incrementa en uno como preparación para la siguiente captación.

Una vez concluido el ciclo de captación, la unidad de control examina el contenido de IR para determinar si contiene un campo de operando que use direccionamiento indirecto. Si es así, se lleva a cabo un *ciclo indirecto*. Tal como muestra la Figura 12.7, se trata de un ciclo sencillo. Los *N* bits más a la derecha de MBR, que contienen la dirección de referencia, se transfieren a MAR.

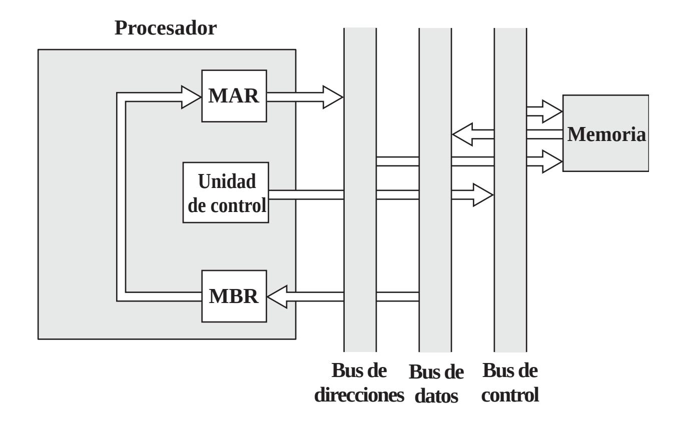


Figura 12.7. Flujo de datos, ciclo indirecto.

Entonces la unidad de control solicita una lectura de memoria para llevar la dirección del operando deseada a MBR.

Los ciclos de captación e indirecto son sencillos y predecibles. El *ciclo de ejecución* adopta muchas formas, ya que depende de cuál de las diversas instrucciones máquina esté en IR. Este ciclo puede implicar transferencias de datos entre registros, lectura o escritura de memoria o E/S, o la utilización de la ALU.

Como los ciclos de captación e indirecto, el *ciclo de interrupción* es simple y predecible (Figura 12.8). El contenido actual de PC tiene que ser guardado para que el procesador pueda reanudar su actividad normal tras la interrupción. Así, el contenido de PC se transfiere a MBR para ser escrito en memoria. La dirección de memoria especial reservada para este propósito se carga en MAR desde la unidad de control. Podría estar, por ejemplo, en un puntero de pila. PC se carga con la dirección de la rutina de interrupción. Como resultado, el siguiente ciclo de instrucción comenzará captando la instrucción oportuna.

12.4. SEGMENTACIÓN DE INSTRUCCIONES

A medida que los computadores evolucionan, se pueden conseguir mayores prestaciones aprovechando los progresos en la tecnología, tales como una circuitería más rápida. Los avances en la organización del procesador también pueden mejorar las prestaciones. Hemos visto ya algunos ejemplos de esto, tales como el empleo de múltiples registros en lugar de un único acumulador y el uso de una memoria caché. Otra aproximación referente a la organización que es bastante común es la segmentación de instrucciones.

ESTRATEGIA DE SEGMENTACIÓN

La segmentación de instrucciones es similar al uso de una cadena de montaje en una fábrica. Una cadena de montaje saca partido del hecho de que el producto pasa a través de varias etapas de producción. Disponiendo el proceso de producción como una cadena de montaje se puede trabajar sobre los productos en varias etapas simultáneamente. Este proceso es conocido como *segmentación de*

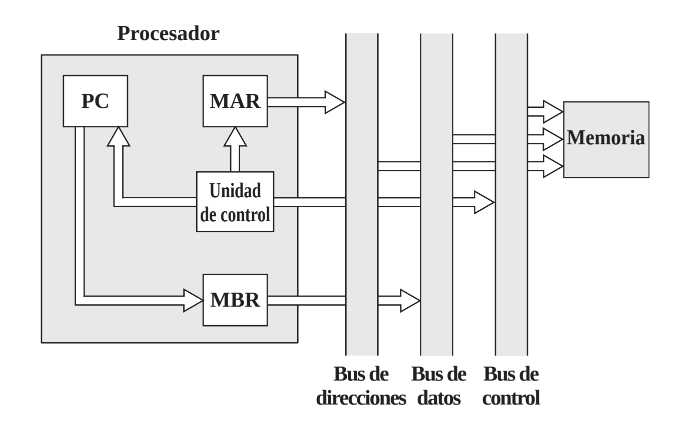


Figura 12.8. Flujo de datos, ciclo de interrupción.

cauce («*pipelining*»), porque, como en una tubería o cauce (*pipeline*), en un extremo se aceptan nuevas entradas antes de que algunas entradas aceptadas con anterioridad aparezcan como salidas en el otro extremo.

Para aplicar este concepto a la ejecución de instrucciones debemos darnos cuenta de que, de hecho, una instrucción tiene varias etapas. La Figura 12.5, por ejemplo, divide el ciclo de instrucción hasta en diez tareas, que tienen lugar secuencialmente. Claramente puede pensarse en la utilización de segmentación.

Como aproximación sencilla, consideremos la subdivisión del procesamiento de una instrucción en dos etapas: captar instrucción y ejecutar instrucción. Hay periodos en la ejecución de una instrucción en los que no se accede a memoria principal. Este tiempo podría utilizarse en captar la siguiente instrucción en paralelo con la ejecución de la actual. La Figura 12.9a representa este planteamiento. El cauce tiene dos etapas independientes. La primera etapa capta una instrucción y la almacena en un *buffer*. Cuando la segunda etapa está libre, la primera le pasa la instrucción almacenada. Mientras que la segunda etapa ejecuta la instrucción, la primera etapa utiliza algún ciclo de memoria no usado para captar y almacenar la siguiente instrucción. Esto se conoce como *prebúsqueda/precaptación de instrucción* («*instruction prefetch*») o *solapamiento de la captación* («*fetch overlap*»).

Debería quedar claro que este proceso acelerará la ejecución de instrucciones. Si las etapas de captación y ejecución fueran de igual duración, el tiempo de ciclo de instrucción se reduciría a la mitad. Sin embargo, si miramos más atentamente a este cauce (Figura 12.9b), veremos que la duplicación de la velocidad de ejecución es poco probable por dos razones:

1. El tiempo de ejecución será generalmente más largo que el tiempo de captación. La ejecución implicará la lectura y almacenamiento de operandos y la realización de alguna operación. Por consiguiente, la etapa de captación puede tener que esperar algún tiempo antes de que pueda vaciar su *buffer*.
2. En una instrucción de salto condicional la dirección de la siguiente instrucción a captar es desconocida. Por tanto, la etapa de captación debe esperar hasta que reciba la dirección de la

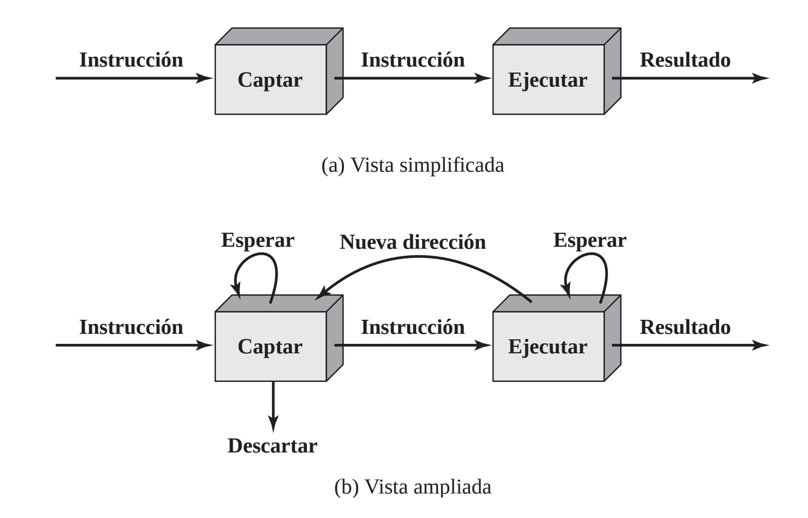


Figura 12.9. Cauce de instrucciones con dos etapas.

siguiente instrucción desde la etapa de ejecución. La etapa de ejecución puede entonces tener que esperar mientras se capta la siguiente instrucción.

La pérdida de tiempo debida a la segunda razón puede reducirse haciendo una estimación. Una regla simple es la siguiente: cuando una instrucción de salto condicional pasa de la etapa de captación a la de ejecución, la etapa de captación capta la instrucción de memoria que sigue a la instrucción de salto. Entonces, si el salto no se produce, no se pierde tiempo. Si el salto se produce, debe desecharse la instrucción captada y captarse una nueva instrucción.

A pesar de que estos factores reducen la efectividad potencial del cauce de dos etapas, se produce cierta aceleración. Para conseguir una mayor aceleración, el cauce debe tener más etapas. Consideremos la siguiente descomposición del procesamiento de una instrucción.

- **Captar instrucción (*Fetch Instruction,* FI):** leer la supuesta siguiente instrucción en un *buffer*.
- **Decodificar instrucción (*Decode Instruction,* DI):** determinar el código de operación y los campos de operando.
- **Calcular operandos (*Calculate Operands,* CO):** calcular la dirección efectiva de cada operando fuente. Esto puede involucrar direccionamiento mediante un desplazamiento, indirecto a través de registro, indirecto, u otras formas de calcular la dirección.
- **Captar operandos (*Fetch Operands,* FO):** captar cada operando que resida en memoria. Los operandos en registros no tienen que ser captados.
- **Ejecutar instrucción (*Execute Instruction,* EI):** realizar la operación indicada y almacenar el resultado, si lo hay, en la posición de operando destino especificada.
- **Escribir operando (*Write Operand,* WO):** almacenar el resultado en memoria.

Con esta descomposición, las diversas etapas tendrán casi igual duración. Por motivos de claridad, asumamos igual duración. En ese supuesto, la Figura 12.10 muestra que un cauce de seis etapas puede reducir el tiempo de ejecución de nueve instrucciones de 54 a catorce unidades de tiempo.

Conviene hacer algunos comentarios. El diagrama supone que cada instrucción recorre las seis etapas del cauce. No siempre se dará este caso. Por ejemplo, una instrucción de carga no necesita la etapa WO. Sin embargo, para simplificar los circuitos del cauce, la temporización se establece suponiendo que cada instrucción requiere las seis etapas. Además, el diagrama supone que todas las etapas pueden funcionar en paralelo. En particular, se supone que no hay conflictos de acceso a memoria. Por ejemplo, las etapas FI, FO y WO requieren un acceso a memoria. El diagrama implica que todos estos accesos pueden tener lugar simultáneamente. La mayoría de los sistemas de memoria no permitirán esto. No obstante, el valor deseado puede estar en caché, o las etapas FO o WO pueden ser nulas. De este modo, la mayor parte del tiempo, los conflictos de memoria no reducirán la velocidad del cauce.

Algunos otros factores contribuyen a limitar la mejora de prestaciones. Si las seis etapas no tienen la misma duración, habrá cierta espera en algunas etapas del cauce, como se discutió antes para el cauce de dos etapas. Otra dificultad es la instrucción de salto condicional, que puede invalidar varias captaciones de instrucciones. Un evento impredecible similar es la llegada de una interrupción. La Figura 12.11 ilustra los efectos del salto condicional, usando el mismo programa que en la

Tiempo →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instrucción 1	FI	DI	CO	FO	EI	WO								
Instrucción 2		FI	DI	CO	FO	EI	WO							
Instrucción 3			FI	DI	CO	FO	EI	WO						
Instrucción 4				FI	DI	CO	FO	EI	WO					
Instrucción 5					FI	DI	CO	FO	EI	WO				
Instrucción 6						FI	DI	CO	FO	EI	WO			
Instrucción 7							FI	DI	CO	FO	EI	WO		
Instrucción 8								FI	DI	CO	FO	EI	WO	
Instrucción 9									FI	DI	CO	FO	EI	WO

Figura 12.10. Diagrama de tiempos del funcionamiento de un cauce de instrucciones.

Figura 12.10. Se supone que la instrucción 3 es un salto condicional a la instrucción 15. Hasta que no se ejecuta la instrucción, no hay forma de saber qué instrucción vendrá a continuación. El cauce, en este ejemplo, simplemente carga la siguiente instrucción secuencialmente (instrucción 4) y continúa. En la Figura 12.10 el salto no se efectúa, y obtenemos el máximo provecho en cuanto a rendimiento de este diseño. En la Figura 12.11 se produce el salto. Este no se determina hasta el final

Tiempo →

Penalización debida al salto ↔

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instrucción 1	FI	DI	CO	FO	EI	WO								
Instrucción 2		FI	DI	CO	FO	EI	WO							
Instrucción 3			FI	DI	CO	FO	EI	WO						
Instrucción 4				FI	DI	CO	FO							
Instrucción 5					FI	DI	CO							
Instrucción 6						FI	DI							
Instrucción 7							FI							
Instrucción 15								FI	DI	CO	FO	EI	WO	
Instrucción 16									FI	DI	CO	FO	EI	WO

Figura 12.11. Efecto de un salto condicional en el funcionamiento de un cauce de instrucciones.

de la unidad de tiempo 7. En ese momento, el cauce debe limpiarse de instrucciones que no son útiles. Durante la unidad de tiempo 8, la instrucción 15 entra en el cauce. Ninguna instrucción termina durante las unidades de tiempo desde la 9 hasta la 12; esta es la penalización en las prestaciones sufrida por no haber podido prever el salto. La Figura 12.12 indica la lógica necesaria para tener en cuenta las bifurcaciones y las interrupciones en la segmentación de cauce.

Se presentan además otros problemas que no aparecían en nuestra organización simple de dos etapas. La etapa CO puede depender del contenido de un registro que podría verse alterado por una instrucción previa que aún esté en el cauce. Podrían ocurrir otros conflictos con registros y con memoria. El sistema tiene que incluir lógica para tener en cuenta este tipo de conflictos.

Para aclarar el funcionamiento del cauce, sería útil considerar una representación alternativa. Las Figuras 12.10 y 12.11 presentan la progresión del tiempo horizontalmente, y cada fila muestra el

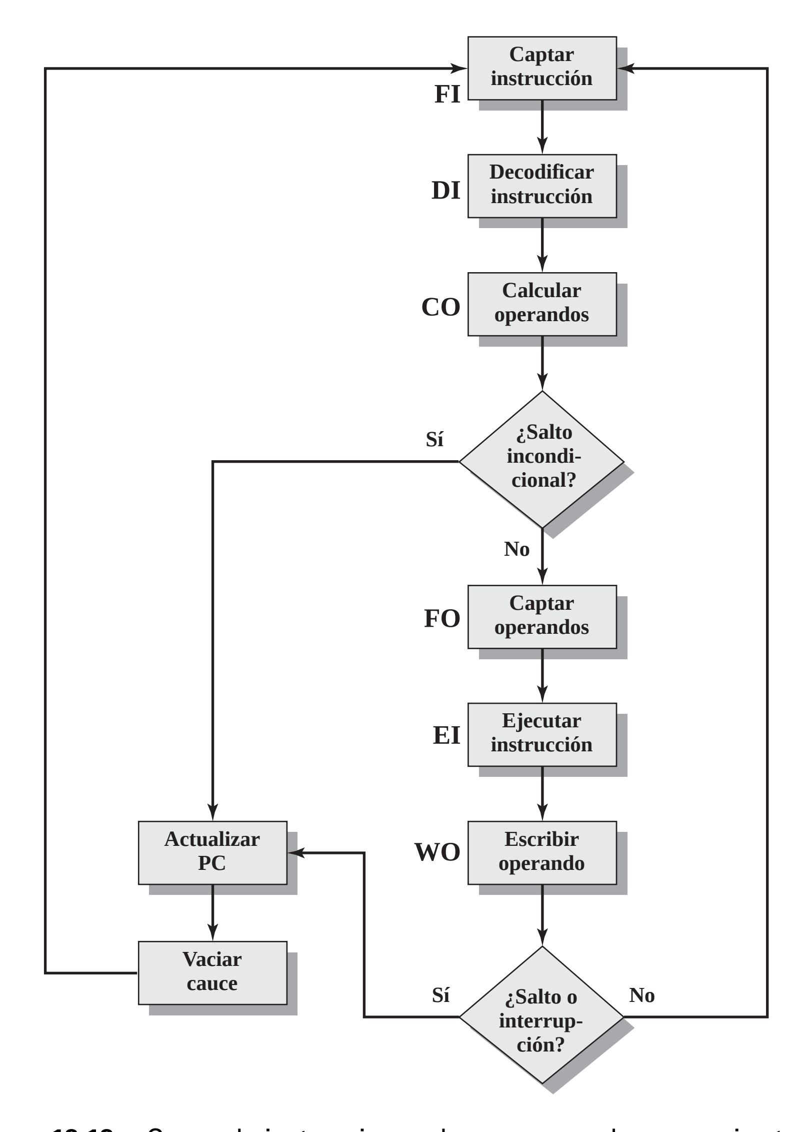


Figura 12.12. Cauce de instrucciones de un procesador, con seis etapas.

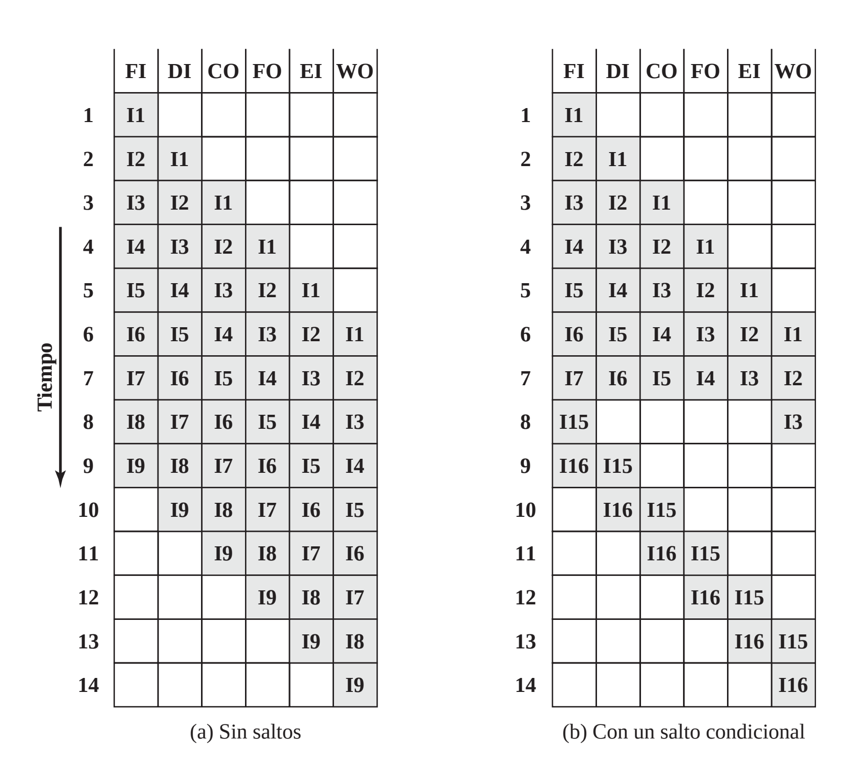


Figura 12.13. Una representación alternativa del cauce.

progreso de una instrucción particular. La Figura 12.13 presenta la misma secuencia de eventos, con el tiempo avanzando verticalmente, y cada fila muestra el estado del cauce en un instante del tiempo. En la Figura 12.13a (que corresponde a la Figura 12.10), el cauce está lleno en el instante 6, existiendo seis instrucciones distintas en varias etapas de ejecución, y permanece lleno hasta el instante 9; suponemos que la instrucción I9 es la última en ejecutarse. En la Figura 12.13b (que corresponde a la Figura 12.11), el cauce está lleno en los instantes 6 y 7. En el instante 7, la instrucción 3 está en la etapa de ejecución y efectúa un salto a la instrucción I15. En ese punto, las instrucciones desde la I4 hasta la I7 son eliminadas del cauce, de forma que en el instante 8, solo hay dos instrucciones en el cauce, I3 e I15.

Según la discusión precedente, podría parecer que cuanto mayor sea el número de etapas en el cauce, más rápida será la velocidad de ejecución. Algunos diseñadores del IBM S/360 observaron dos factores que frustran este aparentemente sencillo patrón de diseño de altas prestaciones [ANDE67a], y que siguen siendo elementos que ha de considerar un diseñador actual:

1. En cada etapa del cauce, hay algún gasto extra debido a la transferencia de datos de buffer a buffer y a la realización de varias funciones de preparación y distribución. Este gasto adicional puede prolongar sensiblemente el tiempo de ejecución total de una instrucción aislada. Esto es importante cuando las instrucciones secuenciales son lógicamente dependientes, bien a causa de un uso abundante de bifurcaciones o bien debido a dependencias de acceso a memoria.
2. La cantidad de lógica de control necesaria para manejar dependencias de memoria y de registros y para optimizar el uso del cauce aumenta enormemente con el número de etapas. Esto

puede llevar a una situación donde la lógica para controlar el paso entre etapas sea más compleja que las propias etapas controladas.

La segmentación de instrucciones es una poderosa técnica para aumentar las prestaciones pero requiere un diseño cuidadoso si se quieren obtener resultados óptimos con una complejidad razonable.

PRESTACIONES DE UN CAUCE SEGMENTADO

En esta subsección desarrollamos algunas medidas sencillas de las prestaciones de un cauce segmentado y del incremento de velocidad relativo (basadas en la discusión de [HWAN93]). El tiempo de ciclo τ de un cauce de instrucciones es el tiempo necesario para que un conjunto de instrucciones avance una etapa a través del cauce; cada columna de las Figuras 12.10 y 12.11 representa un tiempo de ciclo. El tiempo de ciclo puede determinarse como

$$\tau = \max_i[\tau_i] + d = \tau_m + d \qquad 1 \le i \le k$$

donde

τ_i = retardo de tiempo de la circuitería de la i-ésima etapa del cauce
τ_m = máximo retardo de etapa (retardo a través de la etapa que experimenta el mayor retardo)
k = número de etapas del cauce de instrucciones
d = retardo de tiempo de un registro *latch*, necesario para que avancen las señales y datos de una etapa a la siguiente

En general, el retardo de tiempo d es equivalente a un pulso de reloj y $\tau_m \gg d$. Suponga ahora que se procesan n instrucciones, sin saltos. El tiempo total T_{kn} que necesita un cauce de k etapas para ejecutar n instrucciones es

$$T_{kn} = [k + (n - 1)]\tau \tag{12.1}$$

Se requiere un total de k ciclos para completar la ejecución de la primera instrucción, y las n (1 instrucciones restantes requieren $n - 1$ ciclos[2]. Esta ecuación se puede verificar fácilmente en la Figura 12.10. La novena instrucción se completa en el ciclo 14:

$$14 = [6 + (9 - 1)]$$

Considere ahora un procesador con funciones equivalentes pero sin segmentación, y suponga que el tiempo de ciclo de instrucción es $k\tau$. El factor de aceleración para el cauce de instrucciones segmentado comparado con la ejecución sin segmentación se define como

$$S_k = \frac{T_{1,n}}{T_{k,n}} = \frac{nk\tau}{[k + (n - 1)]\tau} = \frac{nk}{k + (n - 1)} \tag{12.2}$$

---

[2] Aquí no estamos siendo muy rigurosos. El tiempo de ciclo solamente igualará el valor máximo de τ cuando todas las etapas estén llenas. En el inicio, el tiempo de ciclo puede ser menor para el primer o los primeros ciclos.

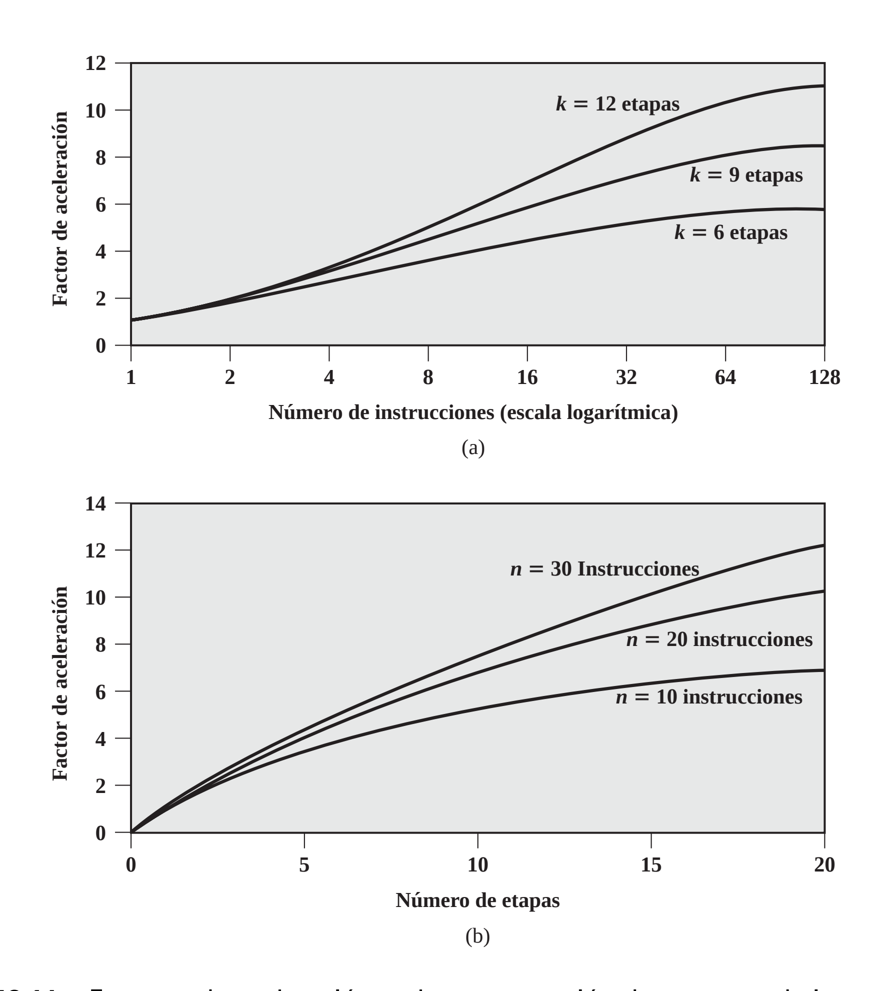


Figura 12.14. Factores de aceleración en la segmentación de un cauce de instrucciones.

La Figura 12.14a representa el factor de aceleración en función del número de instrucciones que se ejecutan sin saltos. Como cabría esperar, en el límite ($n \rightarrow \infty$), la velocidad se incrementa en un factor *k*. La Figura 12.14b muestra el factor de aceleración en función del número de etapas en el cauce de instrucciones[3]. En este caso, el factor de aceleración se aproxima al número de instrucciones que pueden introducirse en el cauce sin saltos. De este modo, cuanto mayor es el número de etapas del cauce, mayor es su potencial para conseguir aceleración. Sin embargo, en la práctica, los beneficios potenciales de etapas adicionales en el cauce se contrarrestan con los incrementos en coste, retardos entre etapas y el hecho de que se encontrarán saltos que requieran vaciar el cauce.

TRATAMIENTO DE SALTOS

Uno de los mayores problemas del diseño de un cauce de instrucciones es asegurar un flujo estable de instrucciones en las etapas iniciales del cauce. El principal obstáculo, como hemos visto, es la

---

[3] Observe que el eje *x* es logarítmico en la Figura 12.14a y lineal en la Figura 12.14b.

instrucción de salto condicional. Hasta que la instrucción no se ejecuta realmente, es imposible determinar si el salto se producirá o no.

Se han considerado varias aproximaciones en el tratamiento de saltos condicionales:

- Flujos múltiples.
- Precaptar el destino del salto.
- Buffer de bucles.
- Predicción de saltos.
- Salto retardado.

Flujos múltiples. Un cauce simple se ve penalizado por las instrucciones de salto porque debe escoger una de las dos instrucciones a captar a continuación y puede hacer una elección equivocada. Un solución burda es duplicar las partes iniciales del cauce y dejar que este capte las dos instrucciones, utilizando los dos caminos. Esta aproximación tiene dos problemas:

- Con cauces múltiples hay retardos debidos a la competencia por el acceso a los registros y a la memoria.
- Pueden entrar en el cauce (en cualquiera de los dos flujos) instrucciones de salto adicionales antes de que se resuelva la decisión del salto original. Cada una de esas instrucciones exige un flujo adicional.

A pesar estos inconvenientes, esta estrategia puede aumentar las prestaciones. El IBM 370/168 y el IBM 3033 son ejemplos de máquinas con dos o más flujos en el cauce.

Precaptar el destino del salto. Cuando se identifica un salto condicional, se precapta la instrucción destino del salto, además de la siguiente a la de salto. Se guarda entonces esta instrucción hasta que se ejecute la instrucción de salto. Si se produce el salto, el destino ya habrá sido precaptado.

El IBM 360/91 usa este método.

***Buffer* de bucles.** Un buffer de bucles es una memoria pequeña de gran velocidad gestionada por la etapa de captación de instrucción del cauce, que contiene, secuencialmente, las *n* instrucciones captadas más recientemente. Si se va a producir un salto, el hardware comprueba en primer lugar si el destino del salto está en el buffer. En ese caso, la siguiente instrucción se capta del buffer. El buffer de bucles tiene tres utilidades:

1. Con el uso de precaptación, el buffer de bucles contendrá algunas instrucciones que secuencialmente están después de la dirección de donde se capta la instrucción actual. De este modo, las instrucciones que se capten secuencialmente estarán disponibles sin necesitar el tiempo de acceso a memoria habitual.
2. Si ocurre un salto a un destino a solo unas pocas posiciones más allá de la dirección de la instrucción de salto, el destino ya estará en el buffer. Esto es útil para el caso bastante común de las secuencias IF-THEN e IF-THEN-ELSE.
3. Esta estrategia se acomoda particularmente bien al tratamiento de bucles, o iteraciones, de ahí el nombre de *buffer de bucles*. Si el buffer de bucles es lo suficientemente grande como

para contener todas las instrucciones de un bucle, entonces esas instrucciones solo han de ser captadas de la memoria una vez, durante la primera iteración. En las siguientes iteraciones, todas las instrucciones necesarias se encuentran ya en el buffer.

El buffer de bucles es similar en principio a una caché de instrucciones. Las diferencias son que el buffer de bucles solo guarda instrucciones consecutivas y que es mucho más pequeño en tamaño, y por tanto de menor coste.

La Figura 12.15 ofrece un ejemplo de un buffer de bucles. Si el buffer contiene 256 bytes, y se usa direccionamiento a bytes, los ocho bits menos significativos se usan para indexar el buffer. Los restantes bits más significativos se examinan para determinar si el destino del salto cae dentro del entorno capturado por el buffer.

Entre las máquinas que usan un buffer de bucles se encuentran algunas de las máquinas CDC (Star-100, 6600, 7600) y el CRAY-1. Una forma especializada de buffer de bucles está disponible en el Motorola 68010, para ejecutar un bucle de tres instrucciones involucrado en la instrucción DBcc (decrementar y saltar si se cumple la condición), (*ver* Problema 12.14). Se mantiene un buffer de tres palabras, y el procesador ejecuta repetidamente estas instrucciones hasta que se satisface la condición del bucle.

Predicción de saltos. Se pueden usar varias técnicas para predecir si un salto se va a producir. Entre las más usuales se encuentran las siguientes:

- Predecir que nunca se salta.
- Predecir que siempre se salta.
- Predecir según el código de operación.
- Conmutador saltar/no saltar.
- Tabla de historia de saltos.

Las tres primeras soluciones son estáticas: no dependen de la historia de la ejecución que haya tenido lugar hasta la instrucción de salto condicional. Las dos últimas aproximaciones son dinámicas: dependen de la historia de la ejecución.

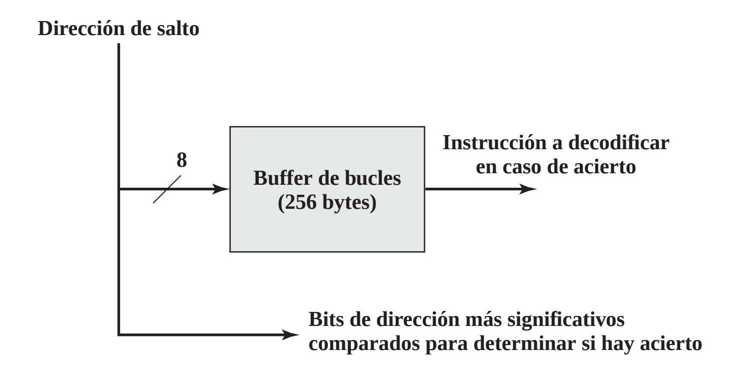


Figura 12.15. Buffer de bucles.

Las dos primeras soluciones son las más sencillas. Una supone que el salto no se producirá y continuará captando instrucciones secuencialmente, y la otra que el salto se producirá y siempre captará la instrucción destino del salto. El 68020 y el VAX 11/780 usan la aproximación de predecir que nunca se salta. El VAX 11/780 además incluye una forma de minimizar el efecto de una decisión errónea. Si la captación de la instrucción que viene después de la de salto causa un fallo de página o una violación de protección, el procesador detiene la prebúsqueda hasta que esté seguro de que la instrucción debe ser captada.

Los estudios que analizan el comportamiento de un programa han mostrado que los saltos condicionales se producen realmente más del cincuenta por ciento de las veces [LILJ88], de forma que si el coste de la prebúsqueda en cada camino es el mismo, precaptar siempre desde la dirección destino del salto debería proporcionar mayores prestaciones que precaptar siempre desde el camino secuencial. Sin embargo, en una máquina que usa paginación, es más probable que cause un fallo de página la prebúsqueda en el destino del salto que la prebúsqueda de la siguiente instrucción secuencial, de manera que esta penalización en las prestaciones debería ser tenida en cuenta. Puede emplearse algún mecanismo que evite dicha penalización.

La última aproximación estática toma la decisión basándose en el código de operación de la instrucción de bifurcación. El procesador supone que el salto se producirá para ciertos códigos de operación de salto y no para otros. [LILJ88] informa sobre tasas de acierto mayores del 75 por ciento con esta estrategia.

Las estrategias de salto dinámicas intentan mejorar la exactitud de la predicción registrando la historia de las instrucciones de salto de un programa. Por ejemplo, a cada instrucción de salto pueden asociarse uno o más bits que reflejen su historia reciente. Estos bits son consultados a modo de conmutador saltar/no saltar que dirige al procesador a tomar una determinada decisión la próxima vez que encuentre la instrucción. Normalmente, estos bits de historia no están asociados a la instrucción en memoria principal. En lugar de ello, se guardan temporalmente en un almacenamiento de alta velocidad. Una posibilidad es asociarlos con cualquier instrucción de salto condicional que esté en caché. Cuando la instrucción en caché es reemplazada, su historia se pierde. Otra posibilidad es mantener una pequeña tabla para las instrucciones de salto ejecutadas recientemente, con uno o más bits de historia en cada elemento de la tabla. El procesador podría acceder a esa tabla asociativamente, como a una caché, o usando los bits de orden inferior de la dirección de la instrucción de salto.

Con un único bit, todo lo que se puede registrar es si la última ejecución de una instrucción dio lugar a un salto o no. Una deficiencia de usar un solo bit se pone de manifiesto en el caso de instrucciones de salto condicional que casi siempre dan lugar a un salto, tales como la instrucción *loop*. Con un único bit de historia, ocurrirá un error en la predicción dos veces en cada uso del bucle: una vez cuando se entra al bucle y otra cuando se sale de él.

Si se usan dos bits, se pueden emplear para registrar el resultado de las dos últimas veces que se ejecutó la instrucción asociada, o para registrar el estado de alguna otra forma. La Figura 12.16 muestra una solución típica (*ver* Problema 12.13 para otras posibilidades). Suponga que el algoritmo comienza en la parte superior izquierda del diagrama de flujo. Siempre que cada instrucción de salto condicional encontrada provoque efectivamente un salto, el proceso de decisión predice que también se producirá el siguiente salto. Si una única predicción es errónea, el algoritmo continúa prediciendo que el siguiente salto se efectuará. Solo si dos instrucciones de salto sucesivas no producen el salto, el algoritmo cambia a la parte derecha del diagrama de flujo. A partir de ahí, el algoritmo predecirá que los saltos no van a producirse hasta que tengan lugar dos saltos seguidos. Por consiguiente, el algoritmo requiere dos predicciones erróneas consecutivas para cambiar la decisión de predicción.

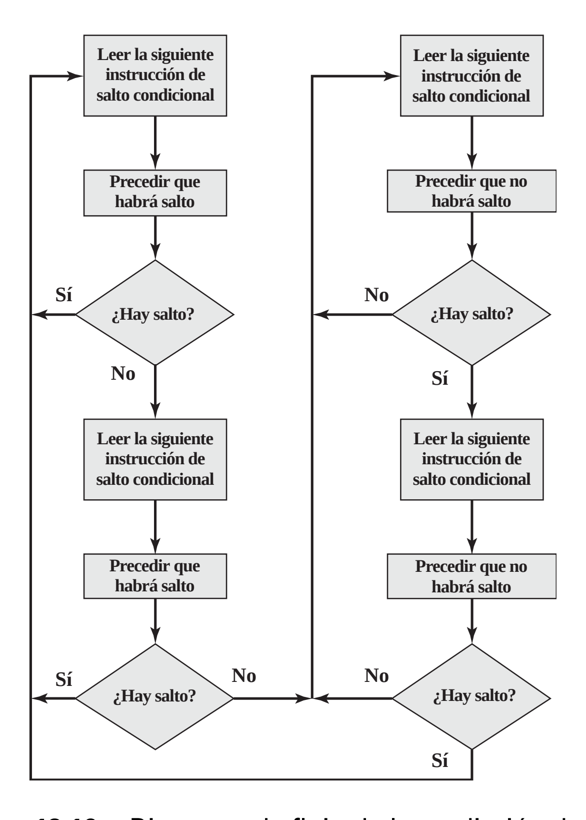


Figura 12.16. Diagrama de flujo de la predicción de saltos.

El proceso de decisión puede representarse de manera más compacta por medio de una máquina de estados finitos, mostrada en la Figura 12.17. La representación de máquina de estados finitos es usada ampliamente en la literatura.

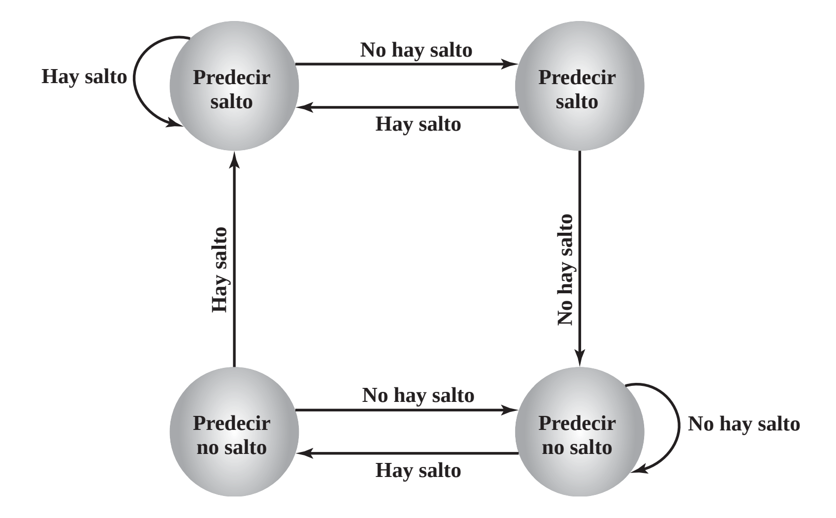


Figura 12.17. Diagrama de estados de la predicción de saltos.

La utilización de bits de historia, como se ha descrito, tiene un inconveniente: si la decisión que se toma es efectuar el salto, la instrucción destino no puede captarse hasta que su dirección, que es un operando de la instrucción de salto condicional, sea decodificada. Se podría lograr una mayor eficiencia si la instrucción captada pudiera iniciarse tan pronto como se tome la decisión de salto. Para ello se debe guardar más información, en lo que se conoce como buffer de destino de saltos, o tabla de historia de saltos.

La tabla de historia de saltos es una pequeña memoria caché asociada a la etapa de captación de instrucción del cauce. Cada elemento de la tabla consta de tres campos: la dirección de una instrucción de salto, un determinado número de bits de historia que guardan el estado de uso de esa instrucción, e información sobre la instrucción destino. En la mayoría de los proyectos e implementaciones, este tercer campo contiene la dirección de la instrucción destino. Otra posibilidad es que el tercer campo contenga la propia instrucción destino. El compromiso es evidente: almacenar la dirección destino conduce a una tabla más pequeña pero a un mayor tiempo de captación de instrucción que el almacenamiento de la instrucción destino [RECH98].

La Figura 12.18 contrasta este esquema frente a una estrategia del tipo predecir que nunca se salta. En la primera estrategia, la etapa de captación de instrucción siempre capta la siguiente dirección secuencial. Si se produce el salto, alguna lógica del procesador lo detecta y ordena que la siguiente instrucción se capte de la dirección destino (además de vaciar el cauce). La tabla de historia de saltos es tratada como una caché. Cada precaptación dispara una búsqueda en la tabla. Si no se encuentra ninguna coincidencia, se usa la dirección siguiente para la captación. Si hay coincidencia, se hace una predicción basada en el estado de la instrucción: a la lógica de selección se suministra bien la dirección secuencial siguiente o bien la dirección destino del salto.

Cuando se ejecuta la instrucción de salto, la etapa de ejecución comunica el resultado a la lógica de la tabla de historia de saltos. El estado de la instrucción se actualiza para reflejar una predicción correcta o incorrecta. Si la predicción es incorrecta, la lógica de selección se desvía hacia la dirección correcta en la siguiente captación. Cuando se encuentra una instrucción de salto condicional que no está en la tabla, se añade a ella y uno de los elementos existentes se desecha, usando uno de los algoritmos de reemplazo de caché estudiados en el Capítulo 4.

Un ejemplo de realización de una tabla de historia de saltos se encuentra en el microprocesador Advanced Micro Device AMD29000.

Salto retardado. Se pueden mejorar las prestaciones de un cauce reordenando automáticamente las instrucciones de un programa, de manera que las instrucciones de salto tengan lugar después de lo realmente deseado. Esta interesante aproximación se examina en el Capítulo 13.

SEGMENTACIÓN DEL INTEL 80486

El 80486 incorpora un cauce de cinco etapas:

- **Captación** (*Fetch*): las instrucciones se captan de la caché o de la memoria externa y se colocan en uno de los dos buffers de prebúsqueda de 16 bytes. El objetivo de la etapa de captación es llenar los buffers de prebúsqueda con nuevos datos tan pronto como los viejos sean tratados por el decodificador de instrucciones. Dado que las instrucciones son de longitud variable (de uno a once bytes sin contar prefijos), el estado del prebuscador relativo a las otras etapas del

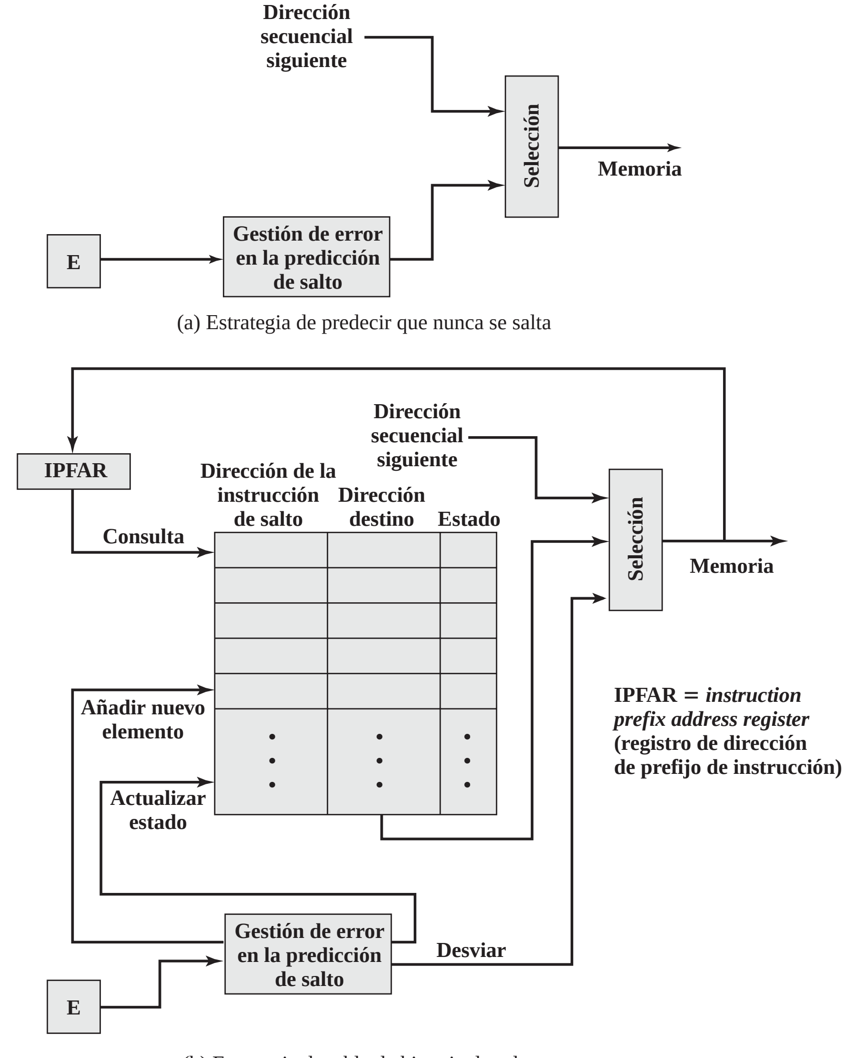


Figura 12.18. Tratamiento de saltos.

cauce varía de instrucción en instrucción. En promedio se captan alrededor de cinco instrucciones con cada carga de 16 bytes [CRAW90]. La etapa de captación opera independientemente de las otras etapas para mantener llenos los buffers de prebúsqueda.

- **Etapa de decodificación 1:** el código de operación y la información referente al modo de direccionamiento se decodifican en la etapa D1. La información necesaria, así como la información sobre la longitud de la instrucción, está contenida a lo sumo en los tres primeros bytes de la instrucción. De ahí que se pasen tres bytes desde los buffers de prebúsqueda a la etapa D1. El decodificador D1 puede entonces indicar a la etapa D2 que capture el resto de la instrucción (desplazamiento y dato inmediato), que no está involucrado en la decodificación de D1.

- **Etapa de decodificación 2:** la etapa D2 convierte cada código de operación en señales de control para la ALU. También controla el cálculo de los modos de direccionamiento más complejos.
- **Ejecución** (***Execute,*** **EX**): esta etapa incluye operaciones de la ALU, acceso a caché y actualización de registros.
- **Escritura** (***Write Back,*** **WB**): esta etapa, cuando es necesaria, actualiza los registros e indicadores de estado modificados durante la etapa de ejecución precedente. Si la instrucción en curso actualiza la memoria, el valor calculado se envía al mismo tiempo a la caché y a los buffers de escritura de la interfaz del bus.

Mediante el uso de dos etapas de decodificación, el cauce puede mantener una productividad cercana a una instrucción por ciclo de reloj. Las instrucciones complejas y los saltos condicionales pueden reducir esta velocidad.

La Figura 12.19 presenta ejemplos de funcionamiento del cauce. La parte (a) muestra que no se introduce ningún retardo en el cauce cuando se necesita un acceso a memoria. No obstante, como muestra la parte (b), puede haber un retardo debido a valores usados para calcular direcciones de memoria. Es decir, si un valor se carga desde memoria a un registro y este se usa como registro base en la siguiente instrucción, el procesador se parará durante un ciclo. En este ejemplo, el procesador accede a la caché en la etapa EX de la primera instrucción y almacena en el registro el valor recuperado durante la etapa WB. Sin embargo, la siguiente instrucción necesita este registro en su etapa D2. Cuando la etapa D2 se alinea con la etapa WB de la instrucción previa, hay caminos que desvían las señales y permiten que la etapa D2 tenga acceso a los mismos datos que está usando la etapa WB para escritura, ahorrando una etapa del cauce.

La Figura 12.19c ilustra la temporización de una instrucción de salto, suponiendo que el salto tiene lugar. La instrucción de comparación actualiza los códigos de condición en la etapa WB, y unos

Fetch	**D1**	**D2**	**EX**	**WB**			**MOV Reg1, Mem1**
	Fetch	**D1**	**D2**	**EX**	**WB**		**MOV Reg1, Reg2**
		Fetch	**D1**	**D2**	**EX**	**WB**	**MOV Mem2, Reg1**

(a) No hay retardo en el cauce debido a la carga de un dato

Fetch	**D1**	**D2**	**EX**	**WB**		**MOV Reg1, Mem1**
	Fetch	**D1**		**D2**	**EX**	**MOV Reg2, (Reg1)**

(b) Un retardo debido a una carga que utiliza un puntero

Fetch	**D1**	**D2**	**EX**	**WB**				**CMP Reg1, Inm**
	Fetch	**D1**	**D2**	**EX**				**Jcc Destino**
				Fetch	**D1**	**D2**	**EX**	**Destino**

(c) Temporización de una instrucción del 80486

Figura 12.19. Ejemplos del cauce de instrucciones del 80486.

caminos de atajo hacen que la etapa EX de la instrucción de salto disponga de ellos en el mismo momento. En paralelo, el procesador ejecuta un ciclo especulativo de captación del destino del salto durante la etapa EX de la instrucción de salto. Si el procesador determina que la condición de salto es falsa, desecha esta precaptación y continúa la ejecución con la siguiente instrucción secuencial (ya captada y decodificada).

12.5. EL PROCESADOR PENTIUM

En la Figura 4.13 se representó una visión de conjunto de la organización del procesador Pentium 4. En esta sección examinaremos algunos detalles.

ORGANIZACIÓN DE LOS REGISTROS

La organización de los registros incluye los siguientes tipos de registros (Tabla 12.2):

- **Generales:** hay ocho registros de uso general de 32 bits (*ver* Figura 12.3c). Pueden ser usados por cualquier tipo de instrucción del Pentium; también pueden contener operandos para cálculos de direcciones. Además, algunos de estos registros pueden servir para usos especiales.

Tabla 12.2. Registros del procesador Pentium.

(a) Unidad de enteros

Tipo	Número	Longitud (bits)	Propósito
Generales	8	32	Registros de usuario de uso general
De segmento	6	16	Contienen selectores de segmento
Indicadores	1	32	Bits de estado y control
Puntero de instrucciones	1	32	Puntero de instrucción

(b) Unidad de coma flotante

Tipo	Número	Longitud (bits)	Propósito
Numérico	8	80	Contienen números en coma flotante
Control	1	16	Bits de control
Estado	1	16	Bits de estado
Palabra de etiquetas	1	16	Especifica los contenidos de los registros numéricos
Puntero de instrucciones	1	48	Apunta a la instrucción interrumpida por una excepción
Puntero de datos	1	48	Apunta al operando interrumpido por una excepción

Por ejemplo, las instrucciones de cadenas usan los contenidos de los registros ECX, ESI y EDI como operandos sin tener que referenciar explícitamente estos registros en la instrucción. Por consiguiente, varias instrucciones pueden codificarse de modo más compacto.

- **De segmento:** los seis registros de segmento de 16 bits contienen selectores de segmento, que indexan tablas de segmentos, como vimos en el Capítulo 8. El registro de segmento de código (*Code Segment*, CS) referencia el segmento que contiene la instrucción que se está ejecutando. El registro de segmento de pila (*Stack Segment*, SS) referencia el segmento que contiene una pila visible por el usuario. Los demás registros de segmento (DS, ES, FS, GS) permiten al usuario referenciar al mismo tiempo hasta cuatro segmentos de datos distintos.
- **Indicadores:** el registro EFLAGS contiene los códigos de condición y diversos bits de modo.
- **Puntero de instrucciones:** contiene la dirección de la instrucción en curso.

Hay también registros dedicados específicamente a la unidad de coma flotante:

- **Numéricos:** cada registro contiene un número en coma flotante de ochenta bits de precisión ampliada. Hay ocho registros que funcionan como una pila, con operaciones *push* y *pop* disponibles en el repertorio de instrucciones.
- **De control:** el registro de control de 16 bits contiene bits que controlan el funcionamiento de la unidad de coma flotante, incluyendo el control del tipo de redondeo; la precisión simple, doble o ampliada; y bits para habilitar e inhabilitar diversas condiciones de excepción.
- **De estado:** el registro de estado de 16 bits contiene bits que reflejan el estado presente de la unidad de coma flotante, incluyendo un puntero al tope de la pila, de tres bits; códigos de condición que informan sobre el resultado de la última operación; e indicadores de excepción.
- **Palabra de etiquetas:** este registro de 16 bits contiene una etiqueta de dos bits para cada registro numérico de coma flotante, que indica la naturaleza de los contenidos del registro correspondiente. Los cuatro valores posibles son válido, cero, especial (NaN, infinito, denormalizado) y vacío. Estas etiquetas permiten a los programas comprobar el contenido de un registro numérico sin tener que realizar una compleja decodificación de los datos que hay en el propio registro. Por ejemplo, cuando se hace un cambio de contexto, el procesador no tiene que guardar ninguno de los registros de coma flotante que están vacíos.

El uso de la mayor parte de los registros anteriores se comprende fácilmente. Detallemos brevemente algunos de los registros.

Registro EFLAGS. El registro EFLAGS (Figura 12.20) indica el estado del procesador y ayuda a controlar su funcionamiento. Incluye los seis códigos de condición definidos en la Tabla 10.9 (acarreo, paridad, acarreo auxiliar, cero, signo, desbordamiento), que informan sobre el resultado de una operación con enteros. Además, en el registro hay bits que podemos denominar de control:

- **Indicador de trampa** (***Trap Flag,*** **TF**): cuando está a uno provoca una interrupción tras la ejecución de cada instrucción. Se usa para depuración.
- **Indicador de habilitación de interrupciones** (***Interrupt Enable Flag,*** **IF**): si está a uno el procesador aceptará las interrupciones externas.

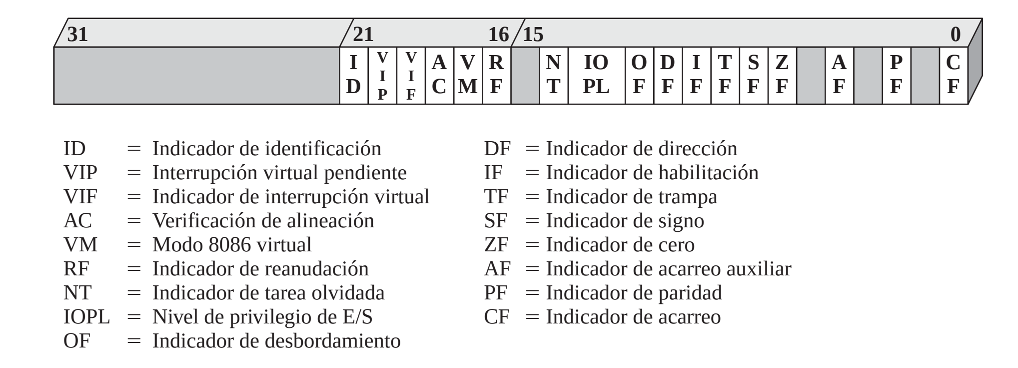


Figura 12.20. Registro EFLAGS del Pentium II.

- **Indicador de dirección (*Direction Flag,* DF):** determina si las instrucciones de procesamiento de cadenas incrementan o decrementan los semi-registros de 16 bits SI y DI (para operaciones de 16 bits) o los registros de 32 bits ESI y EDI (para operaciones de 32 bits).
- **Nivel de privilegio de E/S (*I/O privilege level,* IOPL):** si está a uno hace que el procesador genere una excepción en todos los accesos a dispositivos de E/S cuando funciona en modo protegido.
- **Indicador de reanudación (*Resume Flag,* RF):** permite al programador inhabilitar las excepciones de depuración de manera que la instrucción pueda empezar de nuevo después de una excepción de depuración sin que cause inmediatamente otra excepción de depuración.
- **Verificación de alineación (*Alignment Check,* AC):** controla si una palabra o una doble palabra pueden ser referenciadas en direcciones que no sean múltiplo de dos o de cuatro, respectivamente.
- **Indicador de identificación (*Identification Flag,* ID):** si este bit puede ponerse a uno y borrarse, indica que el procesador soporta la instrucción CPUID. Esta instrucción proporciona información sobre el fabricante, la familia y el modelo.

Además de estos, hay cuatro bits relacionados con el modo de funcionamiento. El indicador de tarea anidada (*nested task,* NT) indica que la tarea actual está anidada con otra tarea cuando el microprocesador funciona en modo protegido. El bit de modo virtual (*virtual mode,* VM) permite al programador habilitar o inhabilitar el modo 8086 virtual, que determina si el procesador funciona o no como una máquina 8086. Los indicadores de interrupción virtual (*virtual interrupt flag,* VIF) e interrupción virtual pendiente (*virtual interrupt pending,* VIP) se usan en entornos multitarea.

Registros de control. El Pentium emplea cuatro registros de control de 32 bits (el registro CR1 no se usa) para controlar varios aspectos del funcionamiento del procesador (Figura 12.21). El registro CR0 contiene indicadores de control del sistema, que controlan modos o indican estados que se aplican generalmente al procesador en lugar de a la ejecución de una tarea individual. Los indicadores son los siguientes:

- **Habilitación de protección (*Protection enable,* PE**): habilita/inhabilita el modo de funcionamiento protegido.

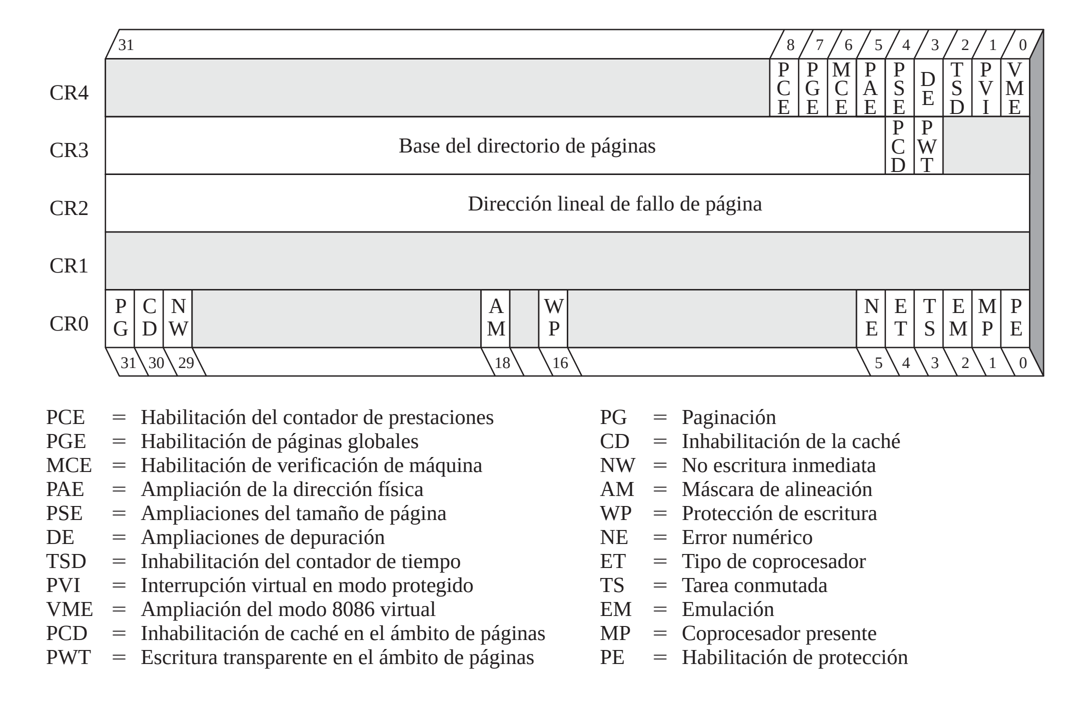


Figura 12.21. Registros de control del Pentium.

- **Coprocesador presente (*Monitor coprocessor,* MP**): solo tiene interés cuando se ejecutan programas de máquinas anteriores al Pentium; indica la presencia de un coprocesador aritmético.
- **Emulación (*Emulation,* EM):** puesto a uno cuando el procesador no tiene una unidad de coma flotante, causa una interrupción cuando se intenta ejecutar instrucciones de coma flotante.
- **Tarea conmutada (*Task Switched,* TS):** indica que el procesador tiene tareas conmutadas.
- **Tipo de coprocesador (*Extension Type,* ET):** no se usa en el Pentium; se usaba para indicar el soporte de instrucciones del coprocesador matemático en máquinas anteriores.
- **Error numérico (*Numeric Error,* NE):** habilita el mecanismo estándar para informar de errores de coma flotante en las líneas de bus externas.
- **Protección de escritura (*Write protect,* WP):** cuando este bit está a cero, un proceso supervisor puede escribir en páginas de nivel de usuario de solo lectura. Esta característica es útil para dar soporte a la creación de procesos en algunos sistemas operativos.
- **Máscara de alineación (*Alignment mask,* AM):** habilita/inhabilita la verificación de alineación.
- **No escritura inmediata (*Not Write through,* NW):** selecciona el modo de funcionamiento de la caché de datos. Cuando este bit está a uno, se impide a la caché de datos realizar operaciones de escritura inmediata.
- **Inhabilitación de caché (*Cache Disable,* CD):** habilita/inhabilita el mecanismo de llenado de la caché interna.
- **Paginación (*Paging,* PG):** habilita/inhabilita la paginación.

Cuando la paginación está habilitada, los registros CR2 y CR3 son válidos. El registro CR2 contiene la dirección lineal de 32 bits de la última página accedida antes de una interrupción de fallo de página. Los veinte bits más a la izquierda de CR3 contienen los veinte bits más significativos de la dirección base del directorio de páginas; el resto de la dirección contiene ceros. Dos bits de CR3 se usan para activar pines de interconexión que controlan el funcionamiento de una caché externa. El bit de inhabilitación de caché en el ámbito de páginas (*page-level caché disable,* PCD) habilita o inhabilita la caché externa, y el bit de escritura transparente en el ámbito de páginas (*page-level writes transparent,* PWT) controla la escritura inmediata en la caché externa.

En CR4 se definen nueve bits de control adicionales:

- **Ampliación del modo 8086 virtual (*Virtual-8086 Mode Extension,* VME):** habilita el soporte del indicador de interrupción virtual en el modo 8086 virtual.
- **Interrupciones virtuales en el modo protegido (*Protected-mode Virtual Interrupts,* PVI):** habilita el soporte del indicador de interrupción virtual en el modo protegido.
- **Inhabilitación del contador de tiempo (*Time Stamp Disable,* TSD):** inhabilita la instrucción de lectura del contador de tiempo (*Read From Time Stamp Counter,* RDTSC) que se usa con objetivos de depuración.
- **Ampliaciones de depuración (*Debugging Extensions,* DE):** habilita puntos de interrupción de E/S; esto permite al procesador interrumpir lecturas y escrituras de E/S.
- **Ampliaciones del tamaño de página (*Page Size Extensions,* PSE):** cuando está a uno, habilita el uso de páginas de 4 MB cuando en el Pentium o de 2 MB en el Pentium Pro o el Pentium II.
- **Ampliación de la dirección física (*Physical Address Extension,* PAE):** habilita las líneas de dirección A35 a A32 cuando un nuevo modo de direccionamiento especial, controlado por el bit PSE, esté habilitado en el Pentium Pro y en posteriores arquitecturas Pentium (Pentium II a Pentium 4).
- **Habilitación de verificación de la máquina (*Machine Check Enable,* MCE):** habilita la interrupción de verificación de la máquina, que tiene lugar cuando ocurre un error de paridad de datos durante un ciclo de lectura del bus o cuando un ciclo del bus no se completa con éxito.
- **Habilitación de páginas globales** (***Page Global Enable,* PGE):** habilita el uso de páginas globales. Cuando PGE = 1 y se produce una conmutación de tarea, todas las entradas de la TLB se vacían, a excepción de aquellas marcadas como globales.
- **Habilitación del contador de prestaciones (*Performance Counter Enable,* PCE):** habilita la ejecución de la instrucción RDPMC (*Read Performance Counter* o lectura de contador de prestaciones) en cualquier nivel de privilegio. Se usan dos contadores de prestaciones para medir la duración de un tipo de evento específico y el número de ocurrencias de un tipo de evento específico.

Registros MMX. En la Sección 10.3 vimos que la capacidad MMX del Pentium utiliza varios tipos de datos de 64 bits. Las instrucciones MMX utilizan campos de direccionamiento de registros de tres bits, de modo que son posibles ocho registros MMX. En realidad, el procesador no incluye registros MMX específicos. En vez de eso, usa una técnica de renombramiento (Figura 12.22). Los registros de coma flotante existentes se usan para almacenar los valores MMX. Concretamente,

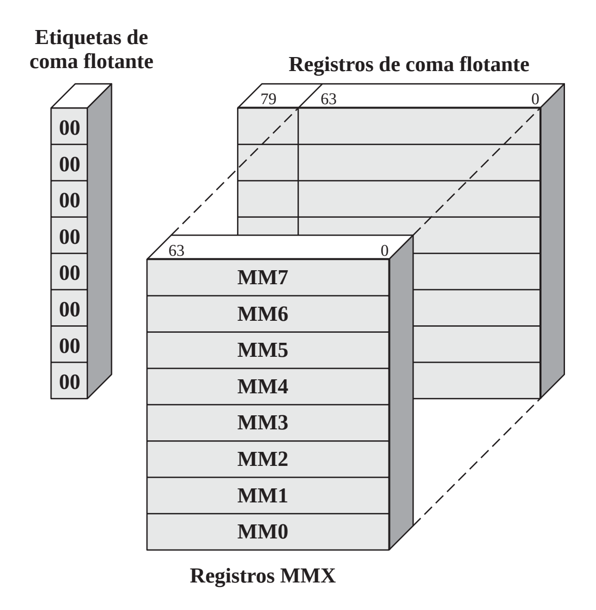


Figura 12.22. Correspondencia entre registros MMX y registros de una coma flotante.

se emplean los 64 bits más bajos (mantisa) de cada registro de coma flotante para formar los ocho registros MMX. De este modo, la arquitectura Pentium existente se amplía con facilidad para admitir la capacidad MMX. El uso MMX de estos registros tiene las siguientes características fundamentales:

- Recordemos que los registros de coma flotante se tratan como una pila en las operaciones de coma flotante. En las operaciones MMX, los mismos registros se acceden directamente.
- La primera vez que una instrucción MMX se ejecuta después de cualquier operación de coma flotante, la palabra de etiquetas de coma flotante se marca como válida. Esto refleja el cambio de funcionamiento como pila a direccionamiento directo a registro.
- La instrucción EMMS (*Empty MMX State*, abandonar estado MMX) fija los bits de la palabra de etiquetas de coma flotante de tal modo que indique que todos los registros se encuentran vacíos. Es importante que el programador coloque esta instrucción al final de un bloque de código MMX para que las operaciones de coma flotante posteriores funcionen correctamente.
- Cuando se escribe un valor en un registro MMX, los bits [79:64] del correspondiente registro de coma flotante (bits de signo y exponente) se ponen todos a uno. Esto fija el valor del registro de coma flotante a NaN (*not a numbe*r, indeterminación) o a infinito cuando se ve como un valor de coma flotante. Ello asegura que un dato MMX no parezca un valor de coma flotante válido.

PROCESAMIENTO DE INTERRUPCIONES

El procesamiento de interrupciones dentro del procesador es un servicio que se proporciona para apoyar al sistema operativo. Permite que un programa de aplicación sea suspendido, para que una gran variedad de causas de interrupción puedan atenderse, y reanudado más tarde.

Interrupciones y excepciones. Hay dos tipos de eventos que hacen que el Pentium suspenda la ejecución del flujo de instrucciones en curso y responda al evento: las interrupciones y las excepciones. En los dos casos, el procesador guarda el contexto del proceso actual y pasa a una rutina predefinida para atender el evento. Una *interrupción* se genera por una señal del hardware, y puede ocurrir en momentos aleatorios durante la ejecución de un programa. Una *excepción* se genera desde el software, y es provocada por la ejecución de una instrucción. Hay dos fuentes de interrupciones y dos fuentes de excepciones:

1. Interrupciones.
   - **Interrupciones enmascarables:** las recibe el procesador por el pin INTR. El procesador no reconoce una interrupción enmascarable a no ser que el indicador de habilitación de interrupciones (IF) esté a uno.
   - **Interrupciones no enmascarables:** las recibe el procesador por al pin NMI. No se puede evitar el reconocimiento de tales interrupciones.

2. Excepciones.
   - **Excepciones detectadas por el procesador:** se producen cuando el procesador encuentra un error mientras intenta ejecutar una instrucción.
   - **Excepciones programadas:** hay instrucciones que generan una excepción (por ejemplo, INTO, INT 3, INT, y BOUND).

Tabla de vectores de interrupción. El procesamiento de las interrupciones en el Pentium usa la tabla de vectores de interrupción. Cada tipo de interrupción tiene asignado un número, que se usa para indexar en la tabla de vectores de interrupción. La tabla contiene 256 vectores de interrupción de 32 bits, cada uno de los cuales es la dirección (segmento y desplazamiento) de la rutina de servicio de interrupción del número de interrupción correspondiente.

La Tabla 12.3 muestra la asignación de números en la tabla de vectores de interrupción; las entradas sombreadas representan interrupciones, mientras que las no sombreadas son excepciones. La interrupción hardware NMI corresponde al vector 2. Las interrupciones hardware INTR tienen asignados números en el rango 32 a 255; cuando se genera una interrupción INTR, debe venir acompañada en el bus con el número de vector de interrupción para esa interrupción. Los números de vectores restantes se usan para las excepciones.

Si hay pendientes más de una excepción o interrupción, el procesador las atiende en un orden previsible. La posición de los números de vector dentro de la tabla no refleja ninguna prioridad. En lugar de ello, la prioridad entre las excepciones e interrupciones se organiza en cinco clases. En orden decreciente de prioridad, son las siguientes:

- **Clase 1:** interceptación en la instrucción previa (vector número 1).
- **Clase 2:** interrupciones externas (2, 32–255).
- **Clase 3:** error captando la instrucción siguiente (3, 14).
- **Clase 4:** error decodificando la instrucción siguiente (6, 7).
- **Clase 5:** error ejecutando una instrucción (0, 4, 5, 8, 10–14, 16, 17).

Tabla 12.3. Tabla de vectores de interrupción y excepción del Pentium.

Número de vector	Descripción
0	Error al dividir; desbordamiento de división o división por cero
1	Excepción de depuración; incluye varios fallos e interceptaciones relacionados con la depuración
2	Interrupción del pin NMI; señal en el pin NMI
3	Punto de parada; causado por la instrucción INT 3, una instrucción de un byte usada para depuración
4	Desbordamiento detectado por INTO; ocurre cuando el procesador ejecuta INTO con el indicador OF a uno
5	Rango de BOUND excedido; la instrucción BOUND compara un registro con unos límites almacenados en memoria y genera una interrupción si el contenido del registro está fuera de los límites
6	Código de operación no definido
7	Dispositivo no disponible; un intento de usar las instrucciones ESC o WAIT da error debido a la falta de un dispositivo externo
8	Doble fallo; dos interrupciones ocurren durante la misma instrucción, y no pueden ser atendidas en serie
9	Reservado
10	Segmento de estado de tarea no válido; el segmento que describe la tarea solicitada no está inicializado o no es válido
11	Segmento no presente; el segmento solicitado no está presente
12	Error en la pila; se ha excedido el límite del segmento de pila o dicho segmento no está presente
13	Protección general; violación de protección que no causa otra excepción (por ejemplo, escribir en un segmento de solo lectura)
14	Fallo de página
15	Reservado
16	Error de coma flotante; generado por una instrucción de aritmética en coma flotante
17	Verificación de alineación; acceso a una palabra almacenada en una dirección de byte impar o a una doble palabra almacenada en una dirección que no sea múltiplo de cuatro
18	Verificación de la máquina; específica para cada modelo
19–31	Reservados
32–255	Vectores de interrupción del usuario; suministrados cuando se activa la señal INTR

No sombreadas: excepciones
Sombreadas: interrupciones

Gestión de interrupciones. De igual modo que las transferencias de control que usan la instrucción CALL, las transferencias a rutinas de gestión de interrupciones también usan la pila del sistema para almacenar el estado del procesador. Cuando se produce una interrupción y es atendida por el procesador, tiene lugar la siguiente secuencia de eventos:

1. Si la transferencia supone un cambio del nivel de privilegio, los contenidos actuales del registro de segmento de pila y del registro puntero de pila ampliado (ESP) se introducen en la pila.
2. El valor actual del registro EFLAGS se introduce en la pila.
3. Los indicadores de interrupciones (IF) y de trampa (TF) se ponen a cero. Ello inhabilita las interrupciones INTR y la interceptación o modo paso a paso.
4. Los contenidos actuales del puntero de segmento de código (CS) y del puntero de instrucciones (IP o EIP) se introducen en la pila.
5. Si la interrupción viene acompañada por un código de error, este se introduce en la pila.
6. Los contenidos del vector de interrupción se captan y se cargan en los registros CS e IP o CS y EIP. La ejecución continúa por la rutina de servicio de interrupción.

Para retornar de una interrupción, la rutina de servicio de interrupción ejecuta una instrucción IRET. Ello provoca que todos los valores almacenados en la pila sean restablecidos; la ejecución se reanuda a partir del punto de interrupción.

12.6. EL PROCESADOR POWERPC

En la Figura 4.14 se mostró una visión de conjunto de la organización del procesador PowerPC. En esta sección examinamos algunos de los detalles de la implementación de 64 bits.

ORGANIZACIÓN DE LOS REGISTROS

La Figura 12.23 representa los registros visibles por el usuario en el PowerPC. La unidad de coma fija incluye los siguientes:

- **Generales:** hay 32 registros de uso general de 64 bits. Se pueden utilizar para cargar, almacenar y manipular operandos de datos y también pueden emplearse para direccionamiento indirecto a través de registro. El registro 0 se trata de una forma algo diferente. Para operaciones de carga y almacenamiento y algunas de las instrucciones de suma, el registro 0 se trata como si tuviera un valor constante cero sin hacer caso de su contenido real.
- **Registro de excepción (*Exception register,* XER):** incluye tres bits que informan sobre excepciones en operaciones aritméticas con enteros. También incluye un campo contador de bytes que se usa como operando en algunas instrucciones con cadenas (Figura 12.24a).

La unidad de coma flotante contiene otros registros visibles por el usuario:

- **Generales:** hay 32 registros de uso general de 64 bits, usados para todas las operaciones de coma flotante.
- **Registro de estado y control de coma flotante (*Floating-Point Status and Control Register,* FPSCR):** este registro de 32 bits contiene bits que controlan el funcionamiento de la unidad de coma flotante, y bits que guardan el estado resultante de operaciones de coma flotante (Tabla 12.4).

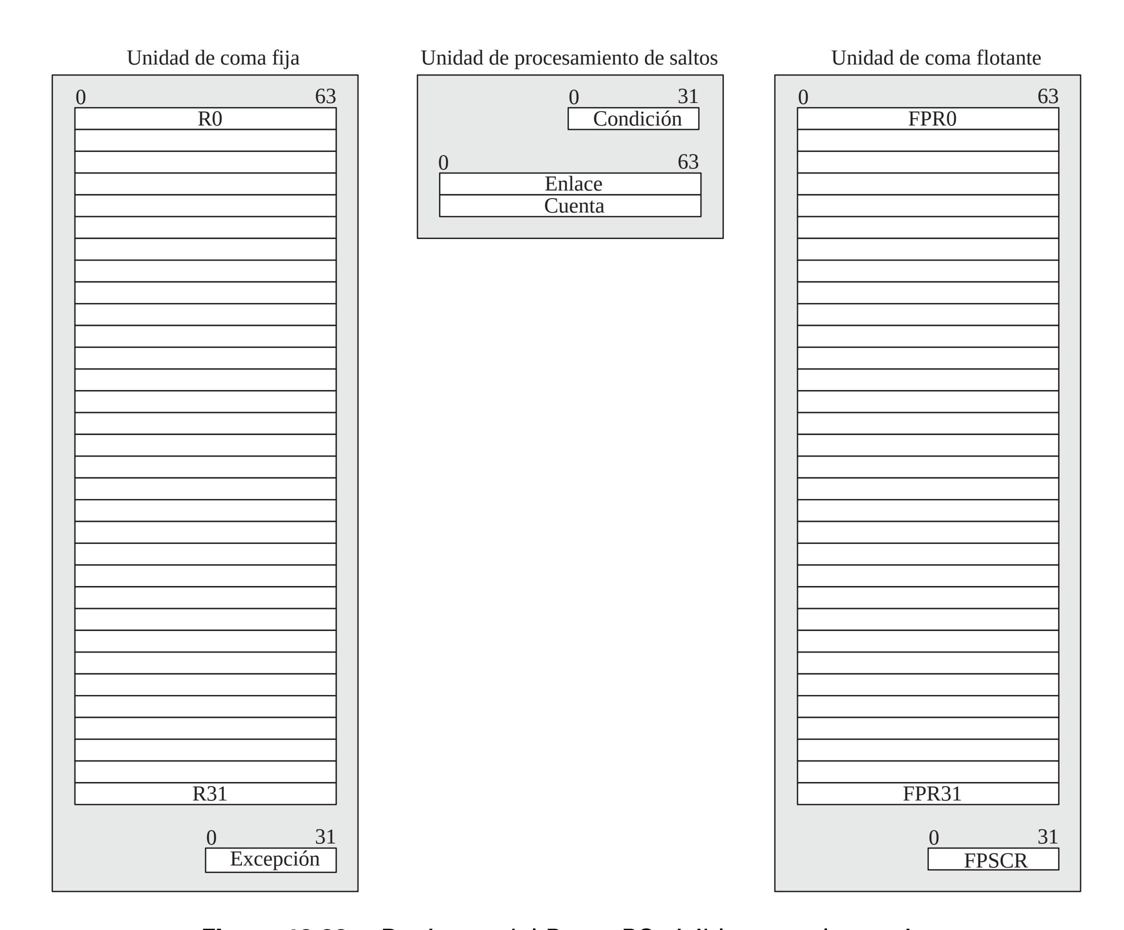
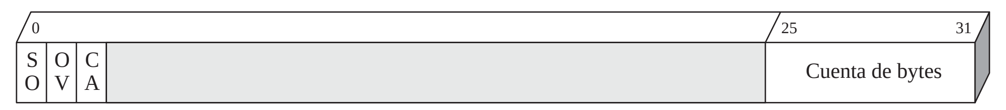


Figura 12.23. Registros del PowerPC visibles por el usuario.

La unidad de procesamiento de saltos contiene los siguientes registros visibles por el usuario:

- **Registro de condición:** consta de ocho campos de código de condición de cuatro bits (Figura 12.24b).
- **Registro de enlace:** el registro de enlace puede usarse en una instrucción de salto condicional para direccionamiento indirecto de la dirección destino. También se usa para el funcionamiento de llamadas y retornos de subrutinas. Si el bit LK en una instrucción de salto condicional está a uno, la dirección siguiente a la instrucción de salto se coloca en el registro de enlace, y se puede usar para un retorno posterior.
- **Cuenta:** el registro de cuenta puede usarse para controlar iteraciones de bucles, como se explicó en el Capítulo 10; este registro se decrementa cada vez que es examinado por una instrucción de salto condicional. Otro uso de este registro es para direccionamiento indirecto de la dirección destino en una instrucción de salto.

Los campos del registro de condición tienen varios usos. Los primeros cuatro bits (CR0) se actualizan en todas las instrucciones aritméticas con enteros para las cuales el bit Rc esté a uno. Como muestra la Tabla 12.5, el campo indica si el resultado de la operación es positivo, negativo o cero. El



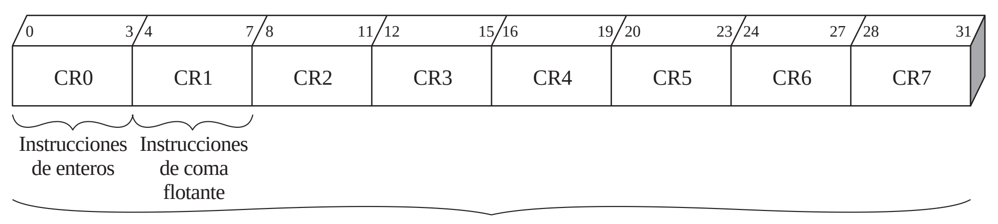
SO = Sumario de desbordamientos (*Summary overflow*): puesto a 1 para indicar que hubo desbordamiento durante la ejecución de una instrucción; permanece a 1 hasta que se borre por software

OV = Desbordamiento (*Overflow*): puesto a 1 para indicar que hubo desbordamiento durante la ejecución de una instrucción; puesto a 0 por la siguiente instrucción si no hay desbordamiento

CA = Acarreo (*Carry*): puesto a 1 para indicar acarreo en el bit 0 durante la ejecución de una instrucción cuenta

Cuenta de bytes = Especifica el número de bytes a transferir por una instrucción indexada de carga/almacenamiento

(a) Registro de excepción de coma fija (XER)



Insrucciones de comparación

(b) Registro de condición

Figura 12.24. Formatos de registros del PowerPC.

Tabla 12.4. Registro de estado y control de coma flotante del PowerPC.

Bit	Definición
0	Sumario de excepciones. Puesto a uno si ocurre alguna excepción; permanece a uno hasta que su puesta a cero mediante software.
1	Sumario de excepciones habilitadas. A uno si ha ocurrido alguna excepción habilitada.
2	Sumario de excepciones de operación no válida. A uno si ha ocurrido una excepción de operación no válida.
3	Excepción de desbordamiento (*overflow*). La magnitud del resultado excede la que se puede representar.
4	Excepción de desbordamiento hacia cero (*underflow*). El resultado es demasiado pequeño para ser normalizado.
5	Excepción de división por cero. El divisor es cero y el dividendo es un número finito distinto de cero.
6	Excepción de inexactitud. El resultado redondeado es distinto del resultado intermedio o se produce desbordamiento con la excepción de desbordamiento inhabilitada.
7:12	Excepción de operación no válida. 7: indica NaN; 8: $(\infty - \infty)$; 9: $(\infty + \infty)$; 10: $(0 \div 0)$; 11: $(\infty \times 0)$; 12: comparación con NaN.

No sombreados: bits de estado
Sombreados: bits de control

(Continúa)

Tabla 12.4. Registro de estado y control de coma flotante del PowerPC *(continuación).*

Bit	Definición
13	Fracción redondeada. El redondeo del resultado incrementó la fracción.
14	Fracción inexacta. El resultado redondeado cambia la fracción u ocurre un desbordamiento con la excepción de desbordamiento inhabilitada.
15:19	Indicadores de resultado. Un código de cinco bits que especifica menor que, mayor que, igual, sin orden, NaN silencioso, $\pm\infty$, $\pm$normalizado, $\pm$denormalizado, ± 0.
20	Reservado.
21:23	Excepción de operación no válida. 21: petición software; 22: raíz cuadrada de un número negativo; 23: conversión entera que involucra un número grande, un infinito, o un NaN.
24	Habilitación de excepción de operación no válida.
25	Habilitación de excepción de desbordamiento (*overflow*).
26	Habilitación de excepción de desbordamiento hacia cero (*underflow*).
27	Habilitación de excepción de división por cero.
28	Habilitación de excepción de inexactitud.
29	Modo no IEEE.
30:31	Control de redondeo. Un código de dos bits especifica al más cercano, a 0, a $+\infty$, o a $-\infty$.

No sombreados: bits de estado
Sombreados: bits de control

Tabla 12.5. Interpretación de los bits del registro de condición.

Posición de bit	CR0 (instrucción entera con Rc = 1)	CR1 (instrucción de coma flotante con Rc = 1)	CRi (instrucción de comparación en coma fija)	CRi (instrucción de comparación en coma flotante)
i	Resultado < 0	Sumario de excepciones	op1 $<$ op2	op1 $<$ op2
$i + 1$	Resultado > 0	Sumario de excepciones habilitadas	op1 $>$ op2	op1 $>$ op2
$i + 2$	Resultado $= 0$	Sumario de excepciones de operación no válida	op1 $=$ op2	op1 $=$ op2
$i + 3$	Sumario de desbordamientos	Excepción de desbordamiento	Sumario de desbordamientos	Sin orden (un operando es un NaN)

cuarto bit es una copia del bit sumario de desbordamientos del XER. El siguiente campo (CR1) se actualiza en todas las instrucciones aritméticas de coma flotante para las cuales el bit Rc esté a uno. En este caso, los cuatro bits son iguales a los cuatro primeros bits del FPSCR (Tabla 12.4). Por último, los ocho campos de condición (CR0 a CR7) se pueden usar con una instrucción de comparación; en cada caso, la identidad del campo se especifica en la propia instrucción. Tanto para las instrucciones

de comparación en coma fija como en coma flotante, los primeros tres bits del campo de condición designado guardan si el primer operando es menor que, mayor que o igual que el segundo operando. El cuarto bit es el bit sumario de desbordamientos para una comparación de coma fija, y un indicador de si no hay posibilidad de ordenación para una comparación en coma flotante.

PROCESAMIENTO DE INTERRUPCIONES

Como en cualquier procesador, el PowerPC incluye un servicio que permite al procesador interrumpir el programa que se está ejecutando para ocuparse de una situación de excepción.

Tipos de interrupciones. Las interrupciones en un PowerPC se clasifican en las que son causadas por alguna condición o evento del sistema y las causadas por la ejecución de una instrucción. La Tabla 12.6 lista las interrupciones que reconoce el PowerPC.

Tabla 12.6. Tabla de interrupciones del PowerPC.

Punto de entrada	Tipo de interrupción	Descripción
00000h	Reservada	
00100h	Reinicio del sistema	Activación desde la lógica externa de las señales de entrada de reinicio físico o lógico del procesador.
00200h	Chequeo de la máquina	Activación de TEA# en el procesador cuando está permitido el reconocimiento de chequeos de la máquina.
00300h	Almacenamiento de datos	Ejemplos: fallo de página de datos, violación de los derechos de acceso en carga/almacenamiento.
00400h	Almacenamiento de instrucciones	Fallo de página de código; intento de captación de instrucción de un segmento de E/S; violación de los derechos de acceso.
00500h	Externa	Activación desde la lógica externa de la señal de entrada al procesador correspondiente a una interrupción externa si está habilitado el reconocimiento de interrupciones externas.
00600h	Alineación	Intento fallido de acceder a memoria debido a un operando mal alineado.
00700h	Programa	Interrupción de coma flotante; el usuario intenta ejecutar una instrucción privilegiada; se ejecutó una instrucción de interceptación con un código de condición especificado; instrucción ilícita.
00800h	Coma flotante no disponible	Intento de ejecutar una instrucción de coma flotante si la unidad de coma flotante se encuentra inhabilitada.
00900h	Decrementador	Agotamiento del registro decrementador si está habilitado el reconocimiento de interrupciones externas.

No sombreadas: interrupciones causadas por la ejecución de una instrucción
Sombreadas: interrupciones no causadas por la ejecución de una instrucción

(Continúa)

Tabla 12.6. Tabla de interrupciones del PowerPC *(continuación).*

Punto de entrada	Tipo de interrupción	Descripción
00A00h	Reservada	
00B00h	Reservada	
00C00h	Llamada al sistema	Ejecución de una instrucción de llamada al sistema.
00D00h	Traza	Interrupción paso a paso o de seguimiento de saltos.
00E00h	Ayuda a coma flotante	Intento de ejecutar operaciones de coma flotante relativamente poco frecuentes o complejas (por ej., operaciones con números denormalizados).
00E10h hasta 00FFFh	Reservada	
01000h hasta 02FFFh	Reservada (depende de cada implementación)	

No sombreadas: interrupciones causadas por la ejecución de una instrucción
Sombreadas: interrupciones no causadas por la ejecución de una instrucción

La mayoría de las interrupciones listadas en la tabla se comprenden fácilmente. Unas pocas justifican el ser comentadas. La interrupción de reinicio del sistema ocurre en el encendido y cuando se pulsa el botón *reset* en la unidad del sistema, y hace que se reinicie el sistema. La interrupción de chequeo de la máquina se ocupa de ciertas anomalías, tales como un error de paridad de caché o una referencia a una posición de memoria inexistente, y puede hacer que el sistema entre en lo que se conoce como estado de parada de chequeo; este estado suspende la ejecución del procesador y congela los contenidos de los registros hasta un reinicio. La ayuda a coma flotante permite al procesador invocar rutinas software para completar operaciones que no puede manejar directamente la unidad de coma flotante, como las que implican números denormalizados o códigos de operación en coma flotante no implementados.

Registro de estado de la máquina (*Machine State Register,* MSR). Para poder interrumpir un programa es fundamental que se tenga la capacidad de recuperar el estado que tenía el procesador en el momento de la interrupción. Este incluye no solo los contenidos de los diversos registros sino también varias condiciones de control relativas a la ejecución. Estas condiciones se almacenan convenientemente en el MSR (Tabla 12.7). De nuevo, algunos bits en este registro necesitan comentarios adicionales.

Cuando el bit de modo de privilegio (bit 49) está a uno, el procesador funciona en un nivel de privilegio de usuario. Solo está disponible un subconjunto del repertorio de instrucciones. Cuando el bit se pone a cero, el procesador funciona en el nivel de privilegio de supervisor. Ello habilita todas las instrucciones y proporciona acceso a ciertos registros del sistema (como el MSR) no accesibles desde el nivel de privilegio de usuario.

Los valores de los dos bits de excepción de coma flotante (bits 52 y 55) definen los tipos de interrupciones que puede generar la unidad de coma flotante. Su interpretación es la siguiente:

FE0	FE1	Interrupciones que se pueden reconocer
0	0	Ninguna
0	1	Imprecisas irrecuperables
1	0	Imprecisas recuperables
1	1	Precisas

Cuando el bit de paso a paso (bit 53) está a uno, el procesador salta al gestor de interrupción de traza después de la finalización con éxito de cada instrucción. Cuando el bit de seguimiento de saltos (bit 54) está a uno, el procesador salta al gestor de interrupción de traza de saltos después de la terminación con éxito de cada instrucción de salto, se produzca el salto o no.

La traducción de direcciones de instrucciones (bit 58) y la traducción de direcciones de datos (bit 59) determinan si se usa direccionamiento real o si la unidad de gestión de memoria lleva a cabo una traducción de las direcciones.

Tabla 12.7. Registro de estado de la máquina en el PowerPC.

Bit	Definición
0	El procesador está en modo de 32 bits/64 bits.
1:44	Reservados.
45	Gestión de energía habilitada/inhabilitada.
46	Depende de la implementación.
47	Define si los gestores de interrupción se ejecutan en modo *big endian* o *little endian.*
48	Interrupción externa habilitada/inhabilitada.
49	Estado privilegiado/no privilegiado.
50	Unidad de coma flotante disponible/no disponible.
51	Interrupciones de chequeo de la máquina habilitadas/no habilitadas.
52	Modo 0 de las excepciones de coma flotante.
53	Ejecución paso a paso habilitada/inhabilitada.
54	Seguimiento de saltos habilitado/inhabilitado.
55	Modo 1 de las excepciones de coma flotante.
56	Reservado.
57	La parte más significativa de la dirección de excepción es 000h/FFFh.
58	Traducción de direcciones de instrucciones activa/inactiva.
59	Traducción de direcciones de datos activa/inactiva.
60:61	Reservados.
62	Interrupción recuperable/irrecuperable.
63	El procesador está en modo *big endian*/*little endian*.

No sombreados: copiados en SRR1
Sombreados: no copiados en SRR1

Gestión de interrupciones. Cuando ocurre una interrupción y es reconocida por el procesador, tiene lugar la siguiente secuencia de eventos:

1. El procesador coloca la dirección de la siguiente instrucción a ejecutar en el registro salvar/restaurar 0 (*Save/Restore Register 0*, SRR0). Esta es la dirección de la propia instrucción que se está ejecutando en ese momento si la interrupción tuvo lugar por un intento fallido de ejecutar dicha instrucción; en cualquier otro caso, es la dirección de la instrucción a ejecutar después de la actual.
2. El procesador copia información sobre el estado de la máquina desde el MSR al registro salvar/restaurar 1 (*Save/Restore Register 1*, SRR1). Se copian los bits no sombreados de la Tabla 12.7. El resto de los bits de SRR1 se cargan con información específica para cada tipo de interrupción.
3. El MSR se fija a un valor, definido por el hardware, específico para cada tipo de interrupción. Para todos los tipos de interrupción, la traducción de direcciones se desactiva y se inhabilitan las interrupciones externas.
4. El procesador transfiere el control al gestor de interrupción apropiado. Las direcciones de los gestores de interrupción están almacenadas en la tabla de interrupciones (Tabla 12.6). La dirección base de la tabla viene determinada por el bit 57 del MSR.

Para retornar de una interrupción, la rutina de servicio de interrupción ejecuta una instrucción RFI (*Return From Interrupt*, retorno de interrupción). Esta hace que los valores de los bits almacenados en SRR1 se recuperen en el MSR. La ejecución se reanuda en la posición almacenada en SRR0.

12.7. LECTURAS RECOMENDADAS

[PATT01] y [MOSH01] cubren excelentemente los asuntos sobre segmentación de cauce estudiados en este capítulo. [HENN91] contiene una discusión detallada sobre segmentación. [SOHI90] ofrece una excelente y detallada discusión sobre los aspectos del diseño hardware implicados en un cauce de instrucciones.

[EVER01] examina la evolución de las estrategias de predicción de saltos. [CRAG92] es un estudio detallado de la predicción de saltos en cauces de instrucciones. [DUBE91] y [LILJ88] examinan varias estrategias de predicción de saltos que se pueden usar para aumentar las prestaciones de la segmentación de instrucciones. [KAEL91] examina la dificultad que introducen en la predicción de saltos las instrucciones cuya dirección de destino es variable.

[BREY03] cubre ampliamente el procesamiento de interrupciones en el Pentium, lo mismo que [SHAN95] para el PowerPC.

BREY03 Brey, B.: *The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4.* Upper Saddle River, NJ. Prentice Hall, 2003.

CRAG92 Cragon, H.: *Branch Strategy Taxonomy and Performance Models.* Los Alamitos, CA. IEEE Computer Society Press, 1992.

DUBE91 Dubey, P. y Flynn, M.: «Branch Strategies: Modeling and Optimization». *IEEE Transactions on Computers,* octubre, 1991.

EVER01 EVERS, M. y YEH, T.: «Understanding Branches and Designing Branch Predictors for Hight-Performance Microprocessors». *Proceedings of the IEEE,* noviembre, 2001.

HENN91 HENNESSY, J. y JOUPPI, N.: «Computer Technology and Architecture: An Evolving Interaction». *Computer,* septiembre, 1991.

KAEL91 KAELI, D. y EMMA, P.: «Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns». *Proceedings, 18th Annual International Symposium on Computer Architecture,* mayo, 1991.

LILJ88 LILJA, D.: «Reducing the Branch Penalty in Pipelined Processors». *Computer,* julio, 1988.

MOSH01 MOSHOVOS, A. y SOHI, G.: «Microarchitectural Innovations:Boosting Microprocessor Performance Beyond Semiconductor Technology Scaling». *Proceedings of the IEEE,* noviembre, 2001.

PATT01 PATT, Y.: «Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution». *Proceedings of the IEEE,* noviembre, 2001.

SHAN95 SHANLEY, T.: *PowerPC System Architecture.* Reading, MA. Addison-Wesley, 1995.

SOHI90 SOHI, G.: «Instruction Issue Logic for High-Performance Interruptable, Multiple Functional Unit, Pipelined Computers». *IEEE Transactions on Computers,* marzo, 1990.

12.8. PALABRAS CLAVE, PREGUNTAS DE REPASO Y PROBLEMAS

PALABRAS CLAVE

cauce de instrucciones ciclo de instrucción código de condiciónindicador	palabra de estado del programa (PSW)	precaptación de instrucciones predicción de saltos salto retardado

PREGUNTAS DE REPASO

12.1. ¿Qué papeles desempeñan en general los registros del procesador?

12.2. ¿Qué tipos de datos son admitidos normalmente por los registro visibles por el usuario?

12.3. ¿Cuál es la función de los códigos de condición?

12.4. ¿Qué es la palabra de estado del programa?

12.5. ¿Por qué es improbable que un cauce de instrucciones de dos etapas reduzca el tiempo de ciclo de instrucción a la mitad, en comparación con un diseño no segmentado?

12.6. Enumere y explique brevemente varias formas de las que un cauce de instrucciones puede ocuparse de las instrucciones de salto condicional.

12.7. ¿Cómo se usan los bits de historia en la predicción de saltos?

PROBLEMAS

12.1. (a) Si la última operación ejecutada en un computador con palabras de ocho bits fue una suma en la que los dos operandos eran 00000010 y 00000011, ¿cuál sería el valor de los siguientes indicadores?

- Acarreo

- Cero
- Desbordamiento
- Signo
- Paridad par
- Acarreo intermedio

(b) ¿Y si los operandos fueran −1 (en complemento a dos) y +1?

12.2. Repita el problema 12.1 para la operación A − B, donde A vale 11110000 y B vale 0010100.

12.3. Un microprocesador trabaja a una frecuencia de reloj de 5 GHz.

(a) ¿Cuál es la duración de un ciclo de reloj?

(b) ¿Cuál es la duración de un tipo particular de instrucción máquina que conste de tres ciclos de reloj?

12.4. Un microprocesador incluye una instrucción capaz de copiar una cadena de bytes de un área de memoria a otra. La captación y la decodificación inicial de la instrucción tardan diez ciclos de reloj. Después, cada byte tarda quince ciclos de reloj en transferirse. El microprocesador funciona a una frecuencia de 5 GHz.

(a) Determine la duración del ciclo de instrucción para el caso de una cadena de 64 bytes.

(b) ¿Cuál es el retardo máximo en atender una interrupción si la instrucción no es interrumpible?

(c) Repita la parte (b) suponiendo que la instrucción puede ser interrumpida al comienzo de cada transferencia de un byte.

12.5. El Intel 8088 consta de una unidad de interfaz del bus (*bus interface unit,* BIU) y de una unidad de ejecución (*execution unit,* EU), las cuales constituyen un cauce de dos etapas. La BIU capta instrucciones en una cola de cuatro bytes. La BIU también interviene en los cálculos de direcciones, capta los operandos y escribe los resultados en la memoria según se lo pide la EU. Si no hay tales peticiones y el bus está libre, la BIU va rellenando los espacios en la cola de instrucciones. Cuando la EU finaliza la ejecución de una instrucción, pasa los posibles resultados a la BIU (con destino en la memoria o la E/S) y pasa a la siguiente instrucción.

(a) Suponga que las tareas realizadas por la BIU y la EU tardan aproximadamente el mismo tiempo. ¿En qué factor incrementa la segmentación las prestaciones del 8088? Ignore el efecto de las instrucciones de salto.

(b) Repita el cálculo suponiendo que la EU tarda el doble que la BIU.

12.6. Suponga que el 8088 esté ejecutando un programa en el cual la probabilidad de un salto es 0,1. Para simplificar, suponga que todas las instrucciones ocupan dos bytes.

(a) ¿Qué fracción de ciclos del bus dedicados a la captación de instrucciones se pierde?

(b) Repita el cálculo para una cola de instrucciones de ocho bytes.

12.7. Considere el diagrama de tiempos de la Figura 12.10. Suponga que hay un cauce de solo dos etapas (captar, ejecutar). Redibuje el diagrama para mostrar cuántas unidades de tiempo se necesitan ahora para cuatro instrucciones.

12.8. Suponga un cauce de cuatro etapas: captar instrucción (FI), decodificar la instrucción y calcular direcciones (DA), captar operando (FO) y ejecutar (EX). Dibuje un diagrama similar a la Figura 12.10 para una secuencia de siete instrucciones, en la cual la tercera instrucción es un salto que tiene lugar y además no hay dependencias de datos.

12.9. Un procesador segmentado tiene una frecuencia de reloj de 2,5 GHz y ejecuta un programa de 1,5 millones de instrucciones. El cauce tiene cinco etapas, y las instrucciones se emiten a una frecuencia de una por ciclo de reloj. Ignore las penalizaciones debidas a las instrucciones de salto y a la ejecución no secuencial.

(a) ¿Cuál es la aceleración de este procesador para este programa en comparación con un procesador no segmentado, bajo las mismas suposiciones de la Sección 12.4?

(b) ¿Cuál es la productividad (en MIPS) del procesador segmentado?

12.10. Un procesador no segmentado tiene una frecuencia de reloj de 2,5 GHz y un número promedio de CPI (ciclos por instrucción) de 4. Una mejora del procesador introduce un cauce de cinco etapas. Sin embargo, debido a retardos internos del cauce, tales como el retardo de *latch*, la frecuencia de reloj del nuevo procesador tiene que reducirse a 2 GHz.

(a) ¿Cuál es la aceleración conseguida en un programa típico?
(b) ¿Cuántos MIPS puede ejecutar cada procesador?

12.11. Considere una secuencia de instrucciones de longitud n que atraviesa un cauce de instrucciones. Sea p la probabilidad de encontrar una instrucción de salto condicional o incondicional, y sea q la probabilidad de que la ejecución de una instrucción de salto I provoque un salto a una dirección no consecutiva. Suponga que cada salto de este tipo requiera vaciar el cauce, destruyendo todo el procesamiento de instrucciones en marcha, cuando I salga de la última etapa. Modifique las Ecuaciones (12.1) y (12.2) de modo que tengan en cuenta estas probabilidades.

12.12. Una limitación de la aproximación de flujos múltiples para tratar los saltos en un cauce es que se encuentren saltos adicionales antes de que se resuelva el primero. Sugiera otras dos limitaciones o desventajas.

12.13. Considere los diagramas de estado de la Figura 12.25.

(a) Describa el funcionamiento de cada uno.
(b) Compárelos con el diagrama de estados de la predicción de saltos de la Sección 12.4. Discuta las ventajas relativas de cada una de las tres aproximaciones para la predicción de saltos.

12.14. Las máquinas Motorola 680x0 incluyen la instrucción «Decrementar y saltar según la condición», que tiene la siguiente forma:

DBcc Dn, desplazamiento

donde cc es una de las condiciones que se pueden examinar, Dn es un registro de uso general, y el desplazamiento especifica la dirección de destino relativa a la dirección actual. La instrucción se puede definir como sigue:

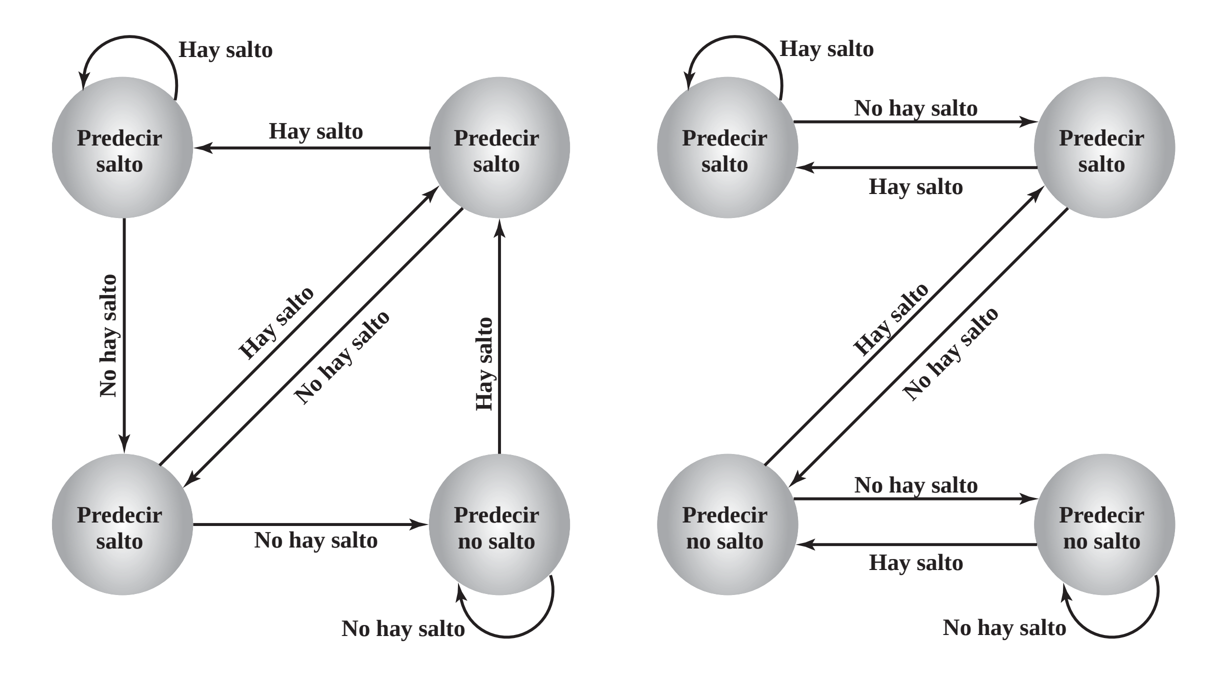


Figura 12.25. Diagramas de estado del Problema 12.13.

if (cc = False)
then begin
  Dn:= (Dn) − 1;
  if Dn ≠ −1 **then** PC:= (PC) + desplazamiento **end**
else PC:= (PC) − 2;

Cuando se ejecuta la instrucción, primero se examina la condición para determinar si se satisface el criterio de terminación del bucle. Si es así, no se realiza ninguna operación y la ejecución continúa secuencialmente por la siguiente instrucción. Si la condición es falsa, se decrementa el registro de datos especificado y se comprueba si es menor que cero. Si lo es, el bucle termina y la ejecución continúa secuencialmente por la siguiente instrucción. En caso contrario, el programa salta a la posición especificada. Considere ahora el siguiente fragmento de programa en lenguaje ensamblador:

```
OTRA_VEZ    CMPM.L    (A0)+, (A1)+
            DBNE      D1, OTRA_VEZ
            NOP
```

Dos cadenas direccionadas por A0 y A1 se comparan para ver si son iguales; los punteros a las cadenas se incrementan en cada referencia. D1 contiene inicialmente el número de palabras largas (4 bytes) a comparar.

(a) Los contenidos iniciales de los registros son A0 = $00004000, A1 = $00005000, y D1 = $000000FF (el $ indica notación hexadecimal). La memoria entre $4000 y $6000 se carga con palabras $AAAA. Si se ejecuta el programa anterior, especifique el número de veces que se ejecuta el bucle DBNE y los contenidos de los tres registros cuando se llega a la instrucción NOP.

(b) Repita el apartado (a), pero suponiendo ahora que la memoria entre $4000 y $4FEE se carga con $0000 y las posiciones entre $5000 y $6000 contienen $AAA.

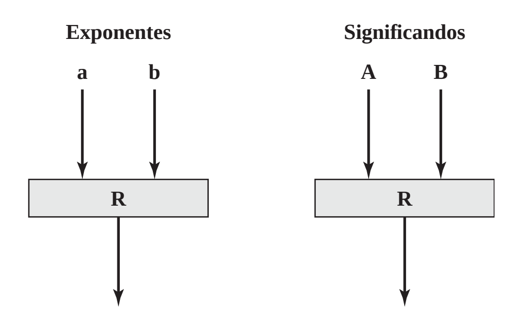
12.15. Vuelva a dibujar la Figura 12.19c suponiendo que el salto condicional no se produce.

12.16. La Tabla 12.8 resume las estadísticas de [MACD84] acerca del comportamiento de los saltos para varias clases de aplicaciones. Con la excepción del comportamiento de los saltos de tipo 1, no hay diferencias perceptibles entre los distintos tipos de aplicaciones. Determine la fracción de saltos que van a la dirección destino del salto.

Tabla 12.8. Comportamiento de los saltos en aplicaciones de ejemplo.

Ocurrencias de los tipos de saltos:			
Tipo 1: saltos	72,5%		
Tipo 2: control de bucles	9,8%		
Tipo 3: llamadas y retornos de procedimientos	17,7%		
Saltos de tipo 1: dónde saltan	**Científicas**	**Comerciales**	**Del sistema**
Incondicionales – 100% van al destino	20%	40%	35%
Condicionales – fueron al destino	43,2%	24,3%	32,5%
Condicionales – no fueron al destino (en línea)	36,8%3	5,7%	32,5%
Saltos de tipo 2 (para todos los entornos)			
Van al destino 91%			
Van en línea 9%			
Saltos de tipo 3			
El 100% van al destino			

12.17. La segmentación puede aplicarse dentro de la ALU para acelerar las operaciones de coma flotante. Considere el caso de la suma y la resta en coma flotante. De manera simplificada, el cauce tendría cuatro etapas: 1. Comparar los exponentes; 2. Escoger el exponente y alinear los significandos o mantisas; 3. Sumar o restar los significandos; 4. Normalizar el resultado. Se puede suponer que el cauce tiene dos hebras paralelas, una para manejar los exponentes y la otra para manejar los significandos, que pueden comenzar del siguiente modo:



En esta figura, las cajas rotuladas como R se refieren al conjunto de registros usados para albergar resultados temporales. Complete el diagrama de bloques que muestre a alto nivel la estructura del cauce segmentado.

CAPÍTULO 13

Computadores de repertorio reducido de instrucciones

PUNTOS CLAVE

- ➤ Los estudios del comportamiento de la ejecución de los programas escritos en lenguajes de alto nivel proporcionaron la orientación para diseñar un nuevo tipo de arquitectura del procesador: el computador de repertorio reducido de instrucciones (*Reduced Instruction Set Computer,* RISC). Las sentencias de asignación predominan, lo que sugiere que las transferencias sencillas de datos deberían optimizarse. Hay también muchas instrucciones IF y LOOP, lo que sugiere que el mecanismo de control de secuencia subyacente ha de ser optimizado para permitir una segmentación eficiente. Los estudios de los patrones de referencia a operandos sugieren que debería ser posible mejorar las prestaciones teniendo guardados un número moderado de operandos en registros.
- ➤ Estos estudios han motivado las características fundamentales de las máquinas RISC: (1) un repertorio de instrucciones limitado y con un formato fijo, (2) un gran número de registros o la utilización de un compilador que optimice el uso de estos, y (3) un énfasis en la optimización del cauce de instrucciones.
- ➤ El repertorio sencillo de instrucciones de un RISC se presta a una segmentación eficiente porque hay menos operaciones llevadas a cabo por cada instrucción y estas son más previsibles. Una arquitectura de repertorio de instrucciones RISC también se presta a la técnica de salto retardado, en la cual las instrucciones de salto se reubican entre otras instrucciones para mejorar la eficiencia del cauce.

Desde el desarrollo del computador de programa almacenado alrededor de 1950, ha habido extraordinariamente pocas innovaciones auténticas en las áreas de organización y arquitectura de computadores. Los siguientes son algunos de los principales avances desde el nacimiento del computador.

- **El concepto de familia:** introducido por IBM en su System/360 en 1964, y seguido poco después por DEC, en su PDP-8. El concepto de familia separa la arquitectura de una máquina de su implementación. Se ofrece un conjunto de computadores, con distintas características en cuanto a precio/prestaciones, que presentan al usuario la misma arquitectura. Las diferencias en precio y prestaciones se deben a las distintas implementaciones de la misma arquitectura.
- **Unidad de control microprogramada:** propuesta por Wilkes en 1951, e introducida por IBM en la línea S/360 en 1964. La microprogramación facilita la tarea de diseñar e implementar la unidad de control y da soporte al concepto de familia.
- **Memoria caché:** introducida comercialmente por primera vez en el IBM S/360 Modelo 85 en 1968. La introducción de este elemento en la jerarquía de memoria mejoró las prestaciones de manera espectacular.
- **Segmentación de cauce:** una manera de introducir paralelismo en la naturaleza esencialmente secuencial de un programa constituido por instrucciones máquina. Dos ejemplos son la segmentación de instrucciones y el procesamiento vectorial.

- **Múltiples procesadores:** esta categoría cubre diferentes organizaciones y objetivos.
- **Arquitectura de computador de repertorio reducido de instrucciones (*Reduced Instruction Set Computer*, RISC):** es el núcleo del presente capítulo.

La arquitectura RISC se aparta de manera drástica de la tendencia histórica en la arquitectura del procesador. Un análisis de la arquitectura RISC involucra la mayoría de los asuntos importantes de organización y arquitectura de computadores.

Aunque los sistemas RISC se han definido y diseñado de diversas formas por parte de diferentes grupos, los elementos principales compartidos por la mayoría de los diseños son:

- Un gran número de registros de uso general, o el uso de tecnología de compiladores para optimizar la utilización de los registros.
- Un repertorio de instrucciones limitado y sencillo.
- Un énfasis en la optimización de la segmentación de instrucciones.

La Tabla 13.1 compara varios sistemas RISC y no RISC.

Comenzamos este capitulo con una revisión breve de algunos resultados relativos a los repertorios de instrucciones, y después examinamos cada uno de los tres asuntos recién mencionados. Sigue a esto una descripción de dos de los diseños RISC mejor documentados.

Tabla 13.1. Características de algunos procesadores CISC, RISC y superescalares.

	Computador de repertorio complejo de instrucciones (CISC)			Computador de repertorio reducido de instrucciones (RISC)		Superescalar		
Característica	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	PowerPC	Ultra SPARC	MIPS R10000
Año de desarrollo	1973	1978	1989	1987	1991	1993	1996	1996
Número de instrucciones	208	303	235	69	94	225	—	—
Tamaño de instrucción (bytes)	2-6	2-57	1-11	4	4	4	4	4
Modos de direccionamiento	4	22	11	1	1	2	1	1
Número de registros de uso general	16	16	8	40-520	32	32	40-520	32
Tamaño de la memoria de control (Kb)	420	480	246	—	—	—	—	—
Tamaño de caché (KB)	64	64	8	32	128	16-32	32	64

13.1. CARACTERÍSTICAS DE LA EJECUCIÓN DE INSTRUCCIONES

Uno de los aspectos relacionados con los computadores que ha evolucionado de manera más visible es el de los lenguajes de programación. Conforme el coste del hardware ha disminuido, el coste relativo del software ha ido creciendo. Junto con esto, la escasez permanente de programadores ha hecho subir los costes del software en términos absolutos. De ahí que el coste principal del ciclo de vida de un sistema sea el software, no el hardware. Otro elemento que se añade al coste y a otros inconvenientes es la falta de fiabilidad: es frecuente que los programas, tanto de sistema como de aplicación, continúen mostrando nuevos errores después de años de funcionamiento.

La respuesta de los investigadores y de la industria fue desarrollar lenguajes de programación de alto nivel más potentes y complejos. Estos lenguajes de alto nivel (*High-Level Languages*, HLL) permiten al programador expresar los algoritmos de manera más concisa, se encargan de gran parte de los pormenores, y a menudo favorecen de manera natural la programación estructurada o el diseño orientado a objetos.

Desgraciadamente, la solución ocasionó otro problema, conocido como el *salto semántico*, la diferencia entre las operaciones que proporcionan los HLL y las que proporciona la arquitectura del computador. Los supuestos síntomas de este salto o hueco incluían ineficiencia de la ejecución, tamaño excesivo del programa en lenguaje máquina, y complejidad de los compiladores. Los diseñadores respondieron con arquitecturas que se proponían cerrar ese hueco. Sus características principales incluyen repertorios de instrucciones grandes, docenas de modos de direccionamiento, y determinadas sentencias HLL implementadas en hardware. Un ejemplo de esto último es la instrucción máquina CASE del VAX. Tales repertorios complejos de instrucciones están pensados para:

- Facilitar el trabajo del escritor de compiladores.
- Mejorar la eficiencia de la ejecución, ya que las secuencias complejas de operaciones pueden implementarse en microcódigo.
- Dar soporte a HLL aun más complejos y sofisticados.

Mientras tanto, se hicieron algunos estudios durante años para determinar las características y patrones de ejecución de las instrucciones máquina generadas por programas escritos en HLL. Los resultados de estos estudios movieron a algunos investigadores a buscar una aproximación diferente: a saber, realizar una arquitectura que diera un soporte más sencillo a los HLL, en lugar de más complejo.

Para comprender la línea de razonamiento de los partidarios de los RISC, comenzamos con una breve revisión de las características de la ejecución de instrucciones. Los aspectos cuyo cálculo tiene interés son los siguientes:

- **Operaciones realizadas:** determinan las funciones que lleva a cabo el procesador y su interacción con la memoria.
- **Operandos usados:** los tipos de operandos y su frecuencia de uso determinan la organización de memoria para almacenarlos y los modos de direccionamiento para acceder a ellos.
- **Secuenciamiento de la ejecución:** determina la organización del control y del cauce segmentado.

En el resto de esta sección, resumimos los resultados de varios estudios sobre programas escritos en lenguaje de alto nivel. Todos los resultados se basan en medidas dinámicas. Es decir, las medidas están tomadas ejecutando el programa y contando el número de veces que alguna característica ha aparecido o una propiedad concreta ha sido cierta. Por contraste, las medidas estáticas solo realizan estas cuentas sobre el texto fuente del programa. Estas no nos dan una información útil sobre las prestaciones, ya que no están ponderadas por el número de veces que se ejecuta cada sentencia.

OPERACIONES

Se han hecho varios estudios para analizar el comportamiento de los programas escritos en HLL. La Tabla 4.8, que se discutió en el Capítulo 4, incluye los principales resultados de varios de estos estudios. Existe una gran concordancia en los resultados de esta mezcla de lenguajes y aplicaciones. Las sentencias de asignación predominan, lo cual indica que el sencillo movimiento de datos tiene mucha importancia. También predominan las sentencias condicionales (IF, LOOP). Estas sentencias se implementan en el lenguaje máquina con alguna instrucción del tipo comparar y saltar. Esto indica que el mecanismo incluido en el repertorio de instrucciones para el control del secuenciamiento es importante.

Estos resultados son instructivos para el diseñador del repertorio de instrucciones máquina, diciéndole qué tipos de sentencias tienen lugar más a menudo y por consiguiente deben ser implementadas de una forma «óptima». No obstante, estos resultados no revelan qué sentencias consumen más tiempo en la ejecución de un programa típico. Es decir, dado un programa en lenguaje máquina compilado, ¿qué sentencias del lenguaje fuente provocan la ejecución de la mayoría de las instrucciones en lenguaje máquina?

Para averiguar este fenómeno subyacente, los programas de Patterson [PATT82a], descritos en el apéndice 4A, se compilaron en VAX, PDP-11 y Motorola 68000 para determinar el número medio de instrucciones máquina y referencias a memoria por cada tipo de sentencia. La segunda y tercera columnas de la Tabla 13.2 muestran la frecuencia relativa de aparición de varias instrucciones de HLL en varios programas; los datos se obtuvieron observando las apariciones en programas en ejecución, en lugar de examinar solo el número de veces que aparecen esas sentencias en el código fuente. Por tanto se trata de estadísticas de frecuencia dinámica. Para obtener los datos de las columnas cuatro y cinco

Tabla 13.2. Frecuencia dinámica relativa ponderada de operaciones en HLL [PATT82a].

	Aparición dinámica		**Instrucciones máquina ponderadas**		**Referencias a memoria ponderadas**	
	Pascal	C	Pascal	C	Pascal	C
ASSIGN	45%	38%	13%	13%	14%	15%
LOOP	5%	3%	42%	32%	33%	26%
CALL	15%	12%	31%	33%	44%	45%
IF	29%	43%	11%	21%	7%	13%
GOTO	—	3%	—	—	—	—
OTRAS	6%	1%	3%	1%	2%	1%

(instrucciones máquina ponderadas), cada valor de la segunda y tercera columnas se multiplicó por el número de instrucciones máquina generadas por el compilador. Estos resultados están además normalizados para que las columnas cuatro y cinco muestren la frecuencia relativa de aparición, ponderada por el número de instrucciones máquina por sentencia de HLL. De un modo parecido, la sexta y séptima columnas se obtienen multiplicando la frecuencia de aparición de cada tipo de sentencia por el número relativo de referencias a memoria originado por esa sentencia. Los datos en las columnas de la cuatro a la siete proporcionan medidas sustitutivas del tiempo real empleado en la ejecución de varios tipos de sentencias. Los resultados indican que la llamada/retorno de procedimientos es la operación que consume más tiempo en los programas típicos escritos en HLL.

El lector debe tener claro el significado de la Tabla 13.2. Esta tabla indica la importancia relativa de varios tipos de sentencias de un HLL, cuando ese HLL se compila para una arquitectura típica con un repertorio de instrucciones actual. Posiblemente, alguna otra arquitectura puede producir diferentes resultados. Sin embargo, este estudio proporciona resultados representativos de las arquitecturas contemporáneas de repertorio complejo de instrucciones (CISC). De ahí que estos resultados puedan servir de orientación a quienes busquen formas más eficientes de dar soporte a los HLL.

OPERANDOS

Se han hecho muchos menos estudios sobre la aparición de los distintos tipos de operandos, a pesar de la importancia de este tema. Hay varios aspectos que son significativos.

El estudio de Patterson ya referenciado [PATT82a] también consideró la frecuencia dinámica de aparición de distintas clases de variables (Tabla 13.3). Los resultados, coherentes para programas en Pascal y en C, muestran que la mayoría de las referencias se hacen a variables escalares simples. Además, más del ochenta por ciento de los datos escalares eran variables locales (al procedimiento). Asimismo, las referencias a matrices/estructuras requieren una referencia previa a su índice o puntero, que nuevamente suele ser un dato escalar local. Por consiguiente, hay un predominio de referencias a operandos escalares, y estos están muy localizados.

El estudio de Patterson examinó el comportamiento dinámico de los programas escritos en HLL, de manera independiente de la arquitectura subyacente. Como se discutió antes, es preciso analizar las arquitecturas reales para examinar el comportamiento del programa con mayor profundidad. Un estudio, [LUND77], examinó dinámicamente las instrucciones del DEC-10 y encontró que cada instrucción referencia en promedio 0,5 operandos de memoria y 1,4 registros. En [HUCK83] se ofrecen resultados semejantes para programas en C, Pascal y FORTRAN ejecutados en los computadores S/370, PDP-11 y VAX. Naturalmente, estas cifras dependen mucho de la arquitectura y del compilador, pero sirven para ilustrar la frecuencia de acceso a los operandos.

Tabla 13.3. Porcentaje dinámico de operandos.

	Pascal	C	Promedio
Constantes enteras	16%	23%	20%
Variables escalares	58%	53%	55%
Matrices/estructuras	26%	24%	25%

Estos últimos estudios reflejan la importancia de una arquitectura que se preste a un rápido acceso a operandos, dado que es una operación que se realiza con mucha frecuencia. El estudio de Patterson indica que un candidato fundamental a optimizar es el mecanismo de almacenamiento y acceso a variables escalares locales.

LLAMADAS A PROCEDIMIENTOS

Hemos visto que las llamadas y retornos de procedimientos constituyen un aspecto importante de los programas en HLL. Los datos (Tabla 13.2) indican que son las operaciones que consumen más tiempo en programas en HLL compilados. Así, será ventajoso considerar formas de implementar estas operaciones eficientemente. Hay dos aspectos importantes: el número de parámetros y variables que maneja un procedimiento, y la profundidad de anidamiento.

El estudio de Tanenbaum [TANE78] encontró que al 98 por ciento de los procedimientos llamados dinámicamente se les pasaban menos de seis argumentos, y que el 92 por ciento de ellos usaban menos de seis variables escalares locales. El equipo de RISC de Berkeley presentó resultados parecidos [KATE83], como se puede ver en la Tabla 13.4. Estos resultados muestran que el número de palabras necesarias por cada activación de un procedimiento no es muy grande. Los resultados expuestos anteriormente indicaban que una gran proporción de referencias a operandos se hacía a variables locales escalares. Estos estudios muestran que tales referencias se limitan, de hecho, a relativamente pocas variables.

El mismo grupo de Berkeley también estudió los patrones seguidos por las llamadas y retornos de procedimientos de programas en HLL. Encontraron que es poco común tener una larga secuencia ininterrumpida de llamadas a procedimientos seguida por la secuencia correspondiente de retornos. Vieron más bien que un programa permanece confinado en una ventana bastante estrecha de profundidad de invocación a procedimientos. Esto lo ilustró la Figura 4.16, que se discutió en el Capítulo 4. Tales resultados refuerzan la conclusión de que las referencias a operandos están muy localizadas.

CONSECUENCIAS

Varios grupos han estudiado resultados como los que se acaban de exponer y han llegado a la conclusión de que intentar realizar una arquitectura de repertorio de instrucciones cercana a los HLL no

Tabla 13.4. Argumentos de procedimientos y variables locales escalares.

Porcentaje de llamadas a procedimientos ejecutadas con	Compilador, intérprete y composición de textos	Pequeños programas no numéricos
>3 argumentos	0-7%	0-5%
>5 argumentos	0-3%	0%
>8 palabras para argumentos y datos escalares locales	1-20%	0-6%
>12 palabras para argumentos y datos escalares locales	1-6%	0-3%

es la estrategia de diseño más efectiva. En su lugar, se puede ofrecer mejor soporte para los HLL optimizando las prestaciones de las características de los programas típicos en HLL que más tiempo consumen.

Partiendo del trabajo de varios investigadores, surgen tres elementos que, por lo general, caracterizan las arquitecturas RISC. El primero de ellos consiste en usar un gran número de registros, o un compilador que optimice su tratamiento. Su finalidad es optimizar las referencias a operandos. Los estudios recién discutidos muestran que hay varias referencias por instrucción de HLL, y que hay una alta proporción de sentencias de transferencia (asignación). Esto, asociado con la localidad y el predominio de las referencias a datos escalares, sugiere que las prestaciones pueden mejorarse reduciendo las referencias a memoria a costa de más referencias a registros. Debido a la localidad de estas referencias, parece práctico un conjunto de registros ampliado.

En segundo lugar, hay que prestar una cuidadosa atención al diseño de los cauces de instrucciones. Debido a la alta proporción de instrucciones de salto condicional y de llamada a procedimientos, un cauce de instrucciones sencillo será ineficiente. Esto se pone de manifiesto como una gran proporción de instrucciones que se precaptan, pero que nunca llegan a ejecutarse.

Por último, es recomendable un repertorio simplificado (reducido) de instrucciones. Este punto no es tan obvio como los otros, pero debería quedar más claro tras la siguiente discusión.

13.2. UTILIZACIÓN DE UN AMPLIO BANCO DE REGISTROS

Los resultados resumidos en la Sección 13.1 indican lo deseable que es un rápido acceso a los operandos. Hemos visto que hay una gran proporción de sentencias de asignación en programas escritos en HLL, y muchas de ellas son de la forma sencilla $A \leftarrow B$. Además hay un número significativo de accesos a operandos por cada sentencia de HLL. Si juntamos estos resultados con el hecho de que la mayoría de los accesos se hacen a datos escalares locales, se puede pensar en una fuerte dependencia del almacenamiento en registros.

La razón de que esté indicado dicho almacenamiento en registros es que estos constituyen el dispositivo de almacenamiento más rápido disponible, más que la memoria principal y que la caché. El banco de registros es pequeño físicamente, está en el mismo chip que la ALU y la unidad de control, y emplea direcciones mucho más cortas que las de la caché y la memoria. Por tanto, se necesita una estrategia que permita que los operandos a los que se acceda con mayor frecuencia se encuentren en registros y se minimicen las operaciones registro-memoria.

Son posibles dos aproximaciones básicas, una basada en software y la otra en hardware. La aproximación por software consiste en confiar al compilador la maximización del uso de los registros. El compilador intentará asignar registros a las variables que se usen más en un periodo de tiempo dado. Esta solución requiere el uso de sofisticados algoritmos de análisis de programas. La aproximación por hardware consiste sencillamente en usar más registros de manera que puedan mantenerse en ellos más variables durante períodos de tiempo más largos.

En esta sección, examinaremos la aproximación por hardware. Ente enfoque se inició en el grupo de RISC de Berkeley [PATT82a]; se usó en el primer producto RISC comercial, el Pyramid [RAGA83]; y se utiliza en la actualidad en la conocida arquitectura SPARC.

VENTANAS DE REGISTROS

A primera vista, el uso de un conjunto amplio de registros debería reducir la necesidad de acceder a memoria. La labor del diseño es organizar los registros de tal modo que se alcance esa meta.

Ya que la mayoría de las referencias a operandos se hacen a datos escalares locales, la solución evidente es almacenar estos en registros, reservando tal vez varios registros para variables globales. El problema es que la definición de *local* varía con cada llamada y retorno de procedimiento, operaciones que suceden con frecuencia. En cada llamada, las variables locales deben guardarse desde los registros en la memoria, para que los registros puedan ser reutilizados por el programa llamado. Además, deben pasarse parámetros. En el retorno, las variables del programa padre tienen que ser restablecidas (cargadas de nuevo en los registros) y hay que devolver los resultados al programa padre.

La solución se basa en otros dos resultados presentados en la Sección 13.1. En primer lugar, un procedimiento típico emplea solo unos pocos parámetros de llamada y variables locales (Tabla 13.4). En segundo lugar, la profundidad de activación de procedimientos fluctúa dentro de un rango relativamente pequeño (Figura 4.16). Para explotar estas propiedades, se usan múltiples conjuntos pequeños de registros, cada uno asignado a un procedimiento distinto. Una llamada a un procedimiento hace que el procesador conmute automáticamente a una ventana de registros distinta de tamaño fijo, en lugar de salvaguardar los registros en memoria. Las ventanas de procedimientos adyacentes están parcialmente solapadas para permitir el paso de parámetros.

El concepto se ilustra en la Figura 13.1. En un momento dado, solo es visible una ventana de registros, y se direcciona como si fuera el único conjunto de registros (por ejemplo, direcciones 0 a $N - 1$). La ventana se divide en tres áreas de tamaño fijo. Los registros de parámetros contienen parámetros pasados al procedimiento actual desde el procedimiento que lo llamó, y los resultados a devolver a este. Los registros locales se usan para variables locales, según la asignación que realice el compilador. Los registros temporales se usan para intercambiar parámetros y resultados con el siguiente nivel más bajo (el procedimiento llamado por el procedimiento en curso). Los registros temporales de un nivel son físicamente los mismos que los registros de parámetros del nivel más bajo adyacente. Este solapamiento posibilita que los parámetros se pasen sin que exista una transferencia de datos real.

Para manejar cualquier patrón posible de llamadas y retornos, el número de ventanas de registros tendría que ser ilimitado. En lugar de eso, las ventanas de registros se pueden usar para contener unas pocas, las más recientes, activaciones de procedimientos. Las activaciones más antiguas han de salvaguardarse en memoria y restaurarse más tarde cuando la profundidad de anidamiento disminuya. De este modo, la organización real del banco de registros es un buffer circular de ventanas solapadas. Dos ejemplos notables de esta solución son la arquitectura SPARC de Sun, descrita en la Sección 13.7, y la arquitectura IA-64 usada en el procesador Itanium de Intel, descrito en el Capítulo 15.

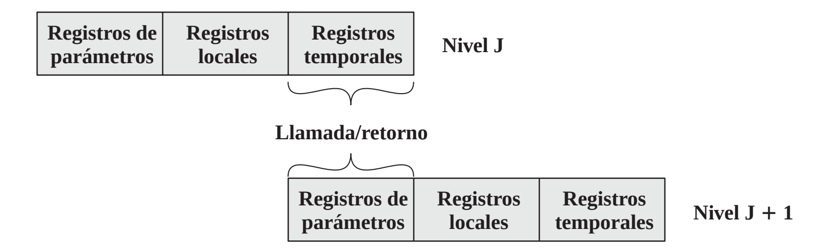


Figura 13.1. Ventanas de registro solapadas.

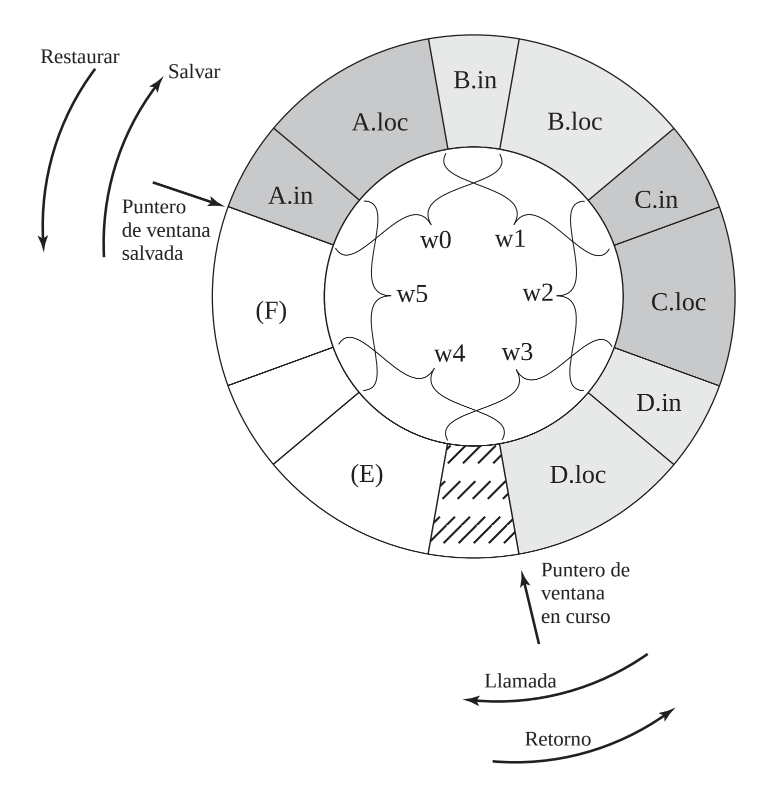


Figura 13.2. Organización de las ventanas solapadas con un buffer circular.

La organización circular se muestra en la Figura 13.2, que representa un buffer circular de seis ventanas. El buffer está lleno hasta una profundidad de 4 (A llamó a B; B llamó a C; y C ha llamado a D) siendo D el procedimiento activo. El puntero de ventana en curso (*Current-Window Pointer,* CWP) apunta a la ventana del procedimiento activo actualmente. Las referencias de una instrucción máquina a registros se transforman con este puntero para determinar el registro físico real. El puntero de ventana salvada (*Saved-Window Pointer,* SWP) identifica la ventana guardada en memoria más recientemente. Si el procedimiento D llama ahora al procedimiento E, los argumentos para E se colocan en los registros temporales de D (el solapamiento entre w3 y w4) y el CWP avanza en una ventana.

Si el procedimiento E hace después una llamada al procedimiento F, esta no puede llevarse a cabo con el estado actual del buffer. Ello se debe a que la ventana de F se solapa con la de A. Si F comienza a cargar sus registros temporales, como preparación para una llamada, ello sobrescribirá los registros de parámetros de A (A.in). Por tanto, cuando al incrementar CWP (módulo 6), este llega a ser igual a SWP, tiene lugar una interrupción, y la ventana de A se guarda. Solo hay que guardar las dos primeras partes (A.in y A.loc). Después se incrementa SWP y la llamada a F sigue adelante. En los retornos ocurre una interrupción similar. Por ejemplo, con posterioridad a la activación de F, cuando B retorna a A, CWP se decrementa y llega a ser igual a SWP. Esto provoca una interrupción cuyo resultado es la restauración de la ventana de A.

De lo anterior, se puede deducir que un banco de registros de N ventanas puede contener solo $N - 1$ activaciones de procedimientos. El valor de N no tiene que ser muy grande. Como mencionamos en el Apéndice 4A, un estudio [TAMI83] encontró que, con ocho ventanas, se requiere una salva-

guarda o restauración sólo en el uno por ciento de las llamadas o retornos. Los computadores RISC de Berkeley usan ocho ventanas de 16 registros cada una. El computador Pyramid emplea 16 ventanas de 32 registros cada una.

VARIABLES GLOBALES

El esquema de ventanas recién descrito proporciona una organización eficiente para el almacenamiento de variables escalares locales en registros. Sin embargo, este esquema no trata la necesidad de almacenar las variables globales, que son las accedidas por más de un procedimiento. Se sugieren dos opciones. La primera es que el compilador asigne posiciones de memoria a las variables declaradas como globales en un HLL, y que todas las instrucciones máquina que referencien esas variables usen operandos referenciados en memoria. Esto es sencillo desde los puntos de vista hardware y software (compilador). No obstante, para variables globales a las que se accede frecuentemente, este esquema es ineficiente.

Una alternativa es incorporar al procesador un conjunto de registros globales. Estos registros serían fijos en cuanto a su número y accesibles por todos los procedimientos. Se puede usar un esquema de numeración unificado para simplificar el formato de instrucción. Por ejemplo, las referencias a los registros desde el 0 hasta el 7 pueden indicar registros globales únicos, y las referencias a los registros del 8 al 31 pueden transformarse para seleccionar los registros físicos de la ventana en curso. Esto conlleva un hardware añadido, que se encarga de adaptar la división en el direccionamiento de los registros. Además, el compilador ha de decidir qué variables globales deben ser asignadas a registros.

UN AMPLIO BANCO DE REGISTROS FRENTE A UNA CACHÉ

El banco de registros organizado en ventanas funciona como un buffer pequeño y rápido que contiene un subconjunto de todas las variables que probablemente se usen más. Desde este punto de vista, el banco de registros se comporta de manera muy similar a una memoria caché, aunque se trate de una memoria mucho más rápida. Surge por tanto la cuestión de si sería más sencillo y mejor usar una caché y un pequeño banco de registros tradicional.

La Tabla 13.5 compara algunas características de las dos soluciones. El banco de registros basado en ventanas contiene todas las variables escalares locales (excepto en el caso poco frecuente de

Tabla 13.5. Características de las organizaciones de un banco de registros amplio y de caché.

Banco de registros amplio	Caché
Todos los datos escalares locales	Datos escalares locales recientemente usados
Variables individuales	Bloques de memoria
Variables globales asignadas por el compilador	Variables globales usadas recientemente
Salvaguarda/restauración basadas en la profundidad de anidamiento	Salvaguarda/restauración basadas en el algoritmo de reemplazo
Direccionamiento a registros	Direccionamiento a memoria

desbordamiento de ventana) de las $N - 1$ activaciones de procedimientos más recientes. La caché contiene una selección de las variables escalares usadas recientemente. El banco de registros debería ahorrar tiempo, ya que guarda todas las variables escalares locales. Por otra parte, la caché puede hacer un uso más eficiente del espacio, ya que reacciona ante las situaciones dinámicamente. Además, las cachés generalmente tratan todas las referencias a memoria del mismo modo, incluyendo las instrucciones y otros tipos de datos. Así, con la caché es posible un ahorro en estas otras áreas y no con un banco de registros.

Un banco de registros puede hacer un uso ineficiente del espacio, puesto que no todos los procedimientos necesitarán todo el espacio de ventana asignado a ellos. Por su parte, la caché adolece de otro tipo de ineficiencia: los datos se leen por bloques. Mientras que el banco de registros contiene solo las variables en uso, la caché lee en un bloque de datos algo o mucho que no se usará.

La caché es capaz de manipular tanto variables globales como locales. Por regla general hay muchos datos escalares globales pero, de ellos, solo unos pocos se usan mucho [KATE83]. Una caché descubrirá dinámicamente esas variables y las almacenará. Si el banco de registros basado en ventanas se complementa con registros globales, también puede contener algunos datos escalares globales. Sin embargo, para un compilador es difícil determinar qué datos globales se usarán mucho.

Con el banco de registros, la transferencia de datos entre registros y memoria viene determinada por la profundidad de anidamiento de los procedimientos. Como esta profundidad normalmente fluctúa en un estrecho margen, el acceso a memoria es relativamente poco frecuente. Casi todas las memorias caché son asociativas por conjuntos con un tamaño de conjunto pequeño. De este modo existe el peligro de que otros datos o instrucciones sobrescriban variables usadas con frecuencia.

Si nos basamos en lo discutido hasta aquí, la elección entre un amplio banco de registros basado en ventanas y una caché no está clara. Existe una característica, no obstante, en la cual la aproximación de los registros es claramente superior y que nos indica que un sistema basado en caché será notablemente más lento. La distinción viene dada por el coste de direccionamiento asociado a cada una de las dos soluciones.

La Figura 13.3 ilustra la diferencia. Para referenciar un dato escalar local en un banco de registros basado en ventanas, se usan un número de registro «virtual» y un número de ventana. Los dos pueden pasar a través de un decodificador relativamente sencillo para seleccionar uno de los registros físicos. Para referenciar una posición de memoria en la caché, hay que generar una dirección de memoria completa. La complejidad de esta operación depende del modo de direccionamiento. En una caché asociativa por conjuntos, una parte de la dirección se usa para leer un número de palabras y etiquetas igual al tamaño del conjunto. Otra parte de la dirección se compara con las etiquetas, y se selecciona una de las palabras leídas. Debe quedar claro que incluso en el caso de que la caché sea tan rápida como el banco de registros, el tiempo de acceso será considerablemente mayor. Por consiguiente, desde el punto de vista de las prestaciones, el banco de registros basado en ventanas es superior para datos escalares locales. Se puede conseguir una mejora adicional en las prestaciones añadiendo una caché solo para instrucciones.

13.3. OPTIMIZACIÓN DE REGISTROS BASADA EN EL COMPILADOR

Supongamos ahora que disponemos de una máquina RISC que contiene únicamente un pequeño número de registros (por ejemplo, 16–32). En tal caso, el uso optimizado de registros es responsabilidad del compilador. Un programa escrito en un lenguaje de alto nivel no tiene, por supuesto, referencias

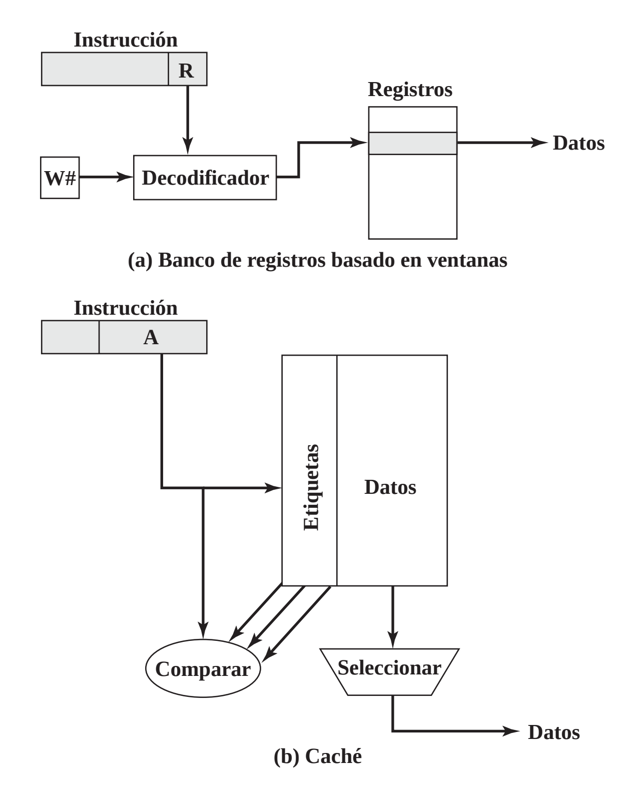


Figura 13.3. Referencia a un dato escalar.

explícitas a los registros. En su lugar, las cantidades usadas en el programa son referidas simbólicamente. El objetivo del compilador es mantener en registros en lugar de en memoria los operandos necesarios para tantos cálculos como sea posible, y minimizar las operaciones de carga y almacenamiento.

Por lo general se sigue el siguiente enfoque. Cada cantidad del programa candidata para residir en un registro se asigna a un registro simbólico o virtual. El compilador entonces asigna el número ilimitado de registros simbólicos a un número fijo de registros reales. Los registros simbólicos cuya utilización no se solape pueden compartir el mismo registro real. Si en una parte concreta del programa hay más cantidades a tratar que registros reales, algunas de las cantidades se asignan a posiciones de memoria. Para colocar temporalmente las cantidades en los registros durante las operaciones de cálculo se usan instrucciones de carga y almacenamiento.

Lo esencial de la labor de optimización es decidir qué cantidades tienen que ser asignadas a registros en un determinado punto del programa. La técnica usada más comúnmente en los compiladores para RISC se conoce como coloreado de grafos, una técnica que procede de la disciplina de topología [CHAI82, CHOW86, COUT86, CHOW90].

El problema del coloreado de grafos es el siguiente. Dado un grafo que consta de nodos y arcos, asignar colores a los nodos de manera que nodos adyacentes tengan colores diferentes, y hacerlo de tal forma que se minimice el número de colores distintos. Este problema se adapta al caso de los compiladores de la siguiente forma. En primer lugar, el programa se analiza para construir un grafo de interferencias entre registros. Los nodos del grafo son los registros simbólicos. Si dos registros simbólicos están «vivos» durante el mismo fragmento de programa, entonces se unen por un arco para

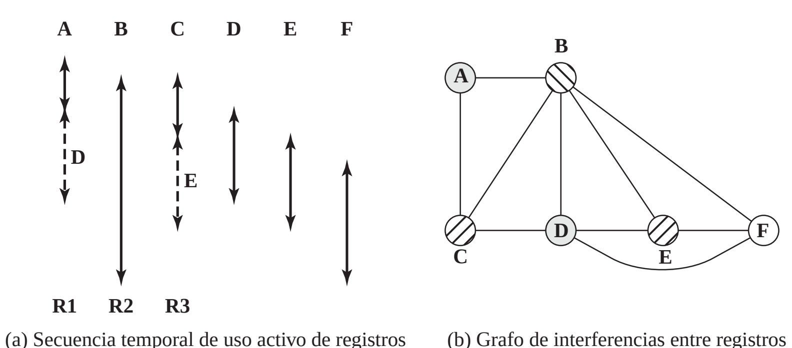
representar su interferencia. Se intenta entonces colorear el grafo con *n* colores, donde *n* es el número de registros. Los nodos que comparten el mismo color pueden asignarse al mismo registro. Si este proceso no tiene éxito completo, los nodos que no se pueden colorear se colocan en memoria, y se tienen que usar cargas y almacenamientos para crear espacio para las cantidades afectadas cuando estas se necesiten.

La Figura 13.4 es un ejemplo sencillo de este proceso. Imaginemos un programa con seis registros simbólicos que tiene que ser compilado para tres registros reales. La Figura 13.4a muestra la secuencia temporal de uso activo de cada registro simbólico, y la parte b muestra el grafo de interferencias entre registros (se usan sombreados y tramas en lugar de colores). Hay que intentar un posible coloreado con tres colores. Un registro simbólico, el F, se queda sin colorear y hay que gestionarlo con el uso de cargas y almacenamientos.

Generalmente existe un compromiso entre el uso de un conjunto de registros grande y la optimización de registros basada en el compilador. Por ejemplo, [BRAD91a] presenta un estudio que modeló una arquitectura RISC con características similares al Motorola 88000 y al MIPS R2000. Los investigadores variaron el número de registros de 16 a 128, y consideraron tanto la utilización de todos los registros para uso general como la separación de registros para enteros y coma flotante. Su estudio mostró que incluso con una optimización de registros sencilla, se saca poco provecho a la utilización de más de 64 registros. Si se usan técnicas de optimización de registros razonablemente sofisticadas, con más de 32 registros solo hay una mejora escasa de las prestaciones. Por último, observaron que con un pequeño número de registros (por ejemplo, 16), una máquina con una organización de registros compartidos funciona más rápido que una con una organización separada. Se pueden extraer conclusiones similares de [HUGU91], que expone un estudio centrado principalmente en la optimización del uso de un número pequeño de registros, en lugar de en la comparación del uso de grandes conjuntos de registros con esfuerzos en la optimización.

13.4. ARQUITECTURA DE REPERTORIO REDUCIDO DE INSTRUCCIONES

En esta sección, consideraremos la motivación y algunas de las características generales de la arquitectura de repertorio reducido de instrucciones. Más adelante, en este mismo capítulo, se estudiarán ejemplos específicos. Comenzamos con una discusión sobre las motivaciones de las arquitecturas contemporáneas de repertorio complejo de instrucciones.



(a) Secuencia temporal de uso activo de registros (b) Grafo de interferencias entre registros

Figura 13.4. La técnica de coloreado de grafos.

¿POR QUÉ CISC?

Hemos mencionado la tendencia hacia repertorios de instrucciones más ricos, que incluyen un número mayor de instrucciones e instrucciones más complejas. Esta tendencia ha sido motivada por dos razones principales: el deseo de simplificar los compiladores y el deseo de mejorar las prestaciones. El cambio de los programadores hacia los lenguajes de alto nivel (HLL) sirvió de base a estas dos razones; los arquitectos intentaron diseñar máquinas que proporcionaran mejor soporte para los HLL.

No es la intención de este capítulo decir que los diseñadores de CISC tomaron la dirección equivocada. Verdaderamente, debido a que la tecnología continúa evolucionando, y a que existen arquitecturas a lo largo de todo un espectro en lugar de en dos categorías puras, es poco probable que alguna vez surja una dicotomía del tipo blanco o negro. Por tanto, los comentarios que siguen solo pretenden señalar algunos de los peligros potenciales de la aproximación CISC y proporcionar cierta comprensión de la motivación de los partidarios de los RISC.

La primera de las razones citadas, la simplificación de los compiladores, parece obvia. La labor del escritor de compiladores es generar una secuencia de instrucciones máquina para cada sentencia de HLL. Si existen instrucciones máquina que se parezcan a sentencias del HLL, la tarea se simplifica. Este razonamiento ha sido puesto en duda por los investigadores de los RISC ([HENN82], [RADI83], [PATT82b]). Estos han encontrado que las instrucciones máquina complejas normalmente son difíciles de aprovechar ya que el compilador debe descubrir aquellos casos que se ajustan perfectamente a la construcción. La tarea de optimizar el código generado para minimizar su tamaño, reducir el número de instrucciones ejecutadas y mejorar la segmentación es mucho más difícil con un repertorio complejo de instrucciones. Como prueba de ello, los estudios citados antes en este capítulo indican que la mayoría de las instrucciones de un programa compilado son comparativamente las más sencillas.

La otra razón importante mencionada es la esperanza de que un CISC produzca programas más pequeños y rápidos. Examinemos los dos aspectos de esta afirmación: que los programas sean más pequeños y que se ejecuten más rápido.

Los programas más pequeños tienen dos ventajas. En primer lugar, como el programa ocupa menos memoria, hay un ahorro de este recurso. Como la memoria es tan barata hoy día, la ventaja potencial ya no es primordial. Tiene mayor importancia el hecho de que programas más pequeños mejoren las prestaciones, lo que ocurre por dos motivos. El primero es que menos instrucciones significa que hay que captar menos bytes de instrucciones. El segundo es que en un entorno paginado, los programas más pequeños ocupan menos páginas, reduciendo los fallos de página.

El problema de esta línea de razonamiento es que está lejos de ser cierto que un programa para un CISC sea más pequeño que el correspondiente programa para un RISC. En muchos casos, el programa para el CISC, expresado en lenguaje máquina simbólico, puede ser *más corto* (esto es, tiene menos instrucciones), pero el número de bits de memoria que ocupa no tiene por qué ser claramente *más pequeño*. La Tabla 13.6 muestra los resultados de tres estudios que compararon el tamaño de programas en C compilados en varias máquinas, incluyendo el RISC I, que tiene una arquitectura de repertorio reducido de instrucciones. Observe que hay poco o ningún ahorro cuando se usa un CISC en lugar de un RISC. Es también interesante advertir que el VAX, que tiene un repertorio de instrucciones mucho más complejo que el PDP-11, logra muy poco ahorro respecto a este último. Los investigadores de IBM confirmaron estos resultados [RADI83], encontrando que el IBM 801 (un RISC) producía código cuyo tamaño era 0,9 veces el del código de un IBM S/370. El estudio utilizó un conjunto de programas en PL/I.

Hay varias razones para estos resultados bastante sorprendentes. Hemos mencionado ya que los compiladores en los CISC tienden a elegir las instrucciones más sencillas, de manera que la concisión

Tabla 13.6. Tamaño del código, relativo al RISC I.

	[PATT82a] 11 programas en C	[KATE83] 12 programas en C	[HEAT84] 5 programas en C
RISC I	1,0	1,0	1,0
VAX-11/780	0,8	0,67	—
M68000	0,9	—	0,9
Z8002	1,2	—	1,12
PDP-11/70	0,9	0,71	—

de las instrucciones complejas raramente entra en juego. También, debido a que hay más instrucciones en un CISC, se necesitan códigos de operación más largos, que dan lugar instrucciones más largas. Por último, los RISC tienden a acentuar las referencias a registros en lugar de a memoria, y las primeras necesitan menos bits. Un ejemplo de este último efecto se discute dentro de poco.

De este modo, las expectativas de que un CISC presente programas más pequeños, con las ventajas relacionadas, pueden no cumplirse. El segundo factor que motivaba repertorios de instrucciones cada vez más complejos era que la ejecución de instrucciones fuera más rápida. Parece tener sentido el que una operación compleja de un HLL se ejecute más rápido como una única instrucción máquina que como una sucesión de instrucciones más primitivas. Sin embargo, debido a la propensión a usar las instrucciones más sencillas, esto puede no ser así. La unidad de control completa debe hacerse más compleja, y/o la memoria de control del microprograma ha de hacerse más grande, para acomodar un repertorio de instrucciones más rico. Cualquiera de los dos factores aumenta el tiempo de ejecución de las instrucciones sencillas.

De hecho, algunos investigadores han encontrado que la aceleración en la ejecución de funciones complejas se debe no tanto a la potencia de las instrucciones máquina complejas como a su residencia en el rápido almacenamiento de control [RADI83]. En efecto, el almacenamiento de control actúa como una caché de instrucciones. De este modo, el arquitecto del hardware está en condiciones de intentar determinar qué subrutinas o funciones se usarán con mayor frecuencia y asignarlas al almacenamiento de control implementándolas en microcódigo. La realidad no es muy alentadora. En los sistemas S/390, instrucciones tales como *Translate* (Traducir) y *Extended-Precision-Floating-Point-Divide* (Dividir en coma flotante con precisión ampliada) residen en almacenamiento de alta velocidad, mientras que la secuencia involucrada en establecer llamadas a procedimientos o en iniciar un gestor de interrupción se encuentran en la memoria principal, que es más lenta.

Por consiguiente, no está nada claro que la tendencia hacia repertorios de instrucciones de complejidad creciente sea apropiada. Ello ha llevado a varios grupos a seguir el camino opuesto.

CARACTERÍSTICAS DE LAS ARQUITECTURAS DE REPERTORIO REDUCIDO DE INSTRUCCIONES

Aunque ha habido diferentes aproximaciones a la arquitectura de repertorio reducido de instrucciones, hay ciertas características comunes a todas ellas:

- Una instrucción por ciclo.
- Operaciones registro a registro.
- Modos de direccionamiento sencillos.
- Formatos de instrucción sencillos.

Vamos a hacer una breve discusión de estas características. Más adelante en este mismo capítulo se examinarán algunos ejemplos concretos.

La primera característica enumerada es que se ejecuta **una instrucción máquina cada ciclo máquina**. Un *ciclo máquina* se define como el tiempo que se tarda en captar dos operandos desde dos registros, realizar una operación de la ALU y almacenar el resultado en un registro. Así, las instrucciones máquina de un RISC no deberían ser más complicadas que las microinstrucciones de las máquinas CISC (tratadas en la Parte Cuatro), y deberían ejecutarse más o menos igual de rápido. Con instrucciones sencillas y de un ciclo, hay poca o ninguna necesidad de microcódigo; las instrucciones máquina pueden estar cableadas. Tales instrucciones deben ejecutarse más rápido que las instrucciones máquina comparables de otras máquinas, ya que no hay que acceder a la memoria de control de microprograma durante la ejecución de la instrucción.

Una segunda característica es que la mayoría de las operaciones deben ser del tipo **registro a registro** con la excepción de sencillas operaciones LOAD y STORE para acceder a memoria. Esta forma de diseño simplifica el repertorio de instrucciones y por tanto la unidad de control. Por ejemplo, un repertorio de instrucciones RISC puede incluir solo una o dos instrucciones ADD (por ejemplo, suma entera y suma con acarreo); el VAX tiene 25 instrucciones ADD diferentes. Otra ventaja es que tal arquitectura fomenta la optimización del uso de registros, ya que los operandos accedidos frecuentemente permanecen en el almacenamiento de alta velocidad.

Este énfasis en las operaciones registro a registro es notable en los diseños RISC. Las máquinas CISC contemporáneas tienen tales instrucciones, pero incluyen también operaciones memoria a memoria y operaciones mixtas registro/memoria. En los años setenta, antes de la aparición de los RISC, se hicieron intentos de comparar estas aproximaciones. La Figura 13.5a muestra el enfoque utilizado. Las arquitecturas hipotéticas se evaluaron según el tamaño del programa y el tráfico de memoria expresado en número de bits. Resultados como este llevaron a un investigador a sugerir que las arquitecturas futuras no deberían contener registros en absoluto [MYER78]. Uno se pregunta qué habría pensado, en su momento, de la máquina RISC comercializada por Pyramid, ¡con nada menos que 528 registros!

Lo que se pasó por alto en estos estudios fue un reconocimiento de que hay un acceso frecuente a un número pequeño de datos escalares locales y de que, con un amplio banco de registros o un compilador con capacidad de optimización, la mayoría de los operandos pueden permanecer en registros durante largos periodos de tiempo. Por eso, la Figura 13.5b puede ser una comparación más justa.

Una tercera característica es el uso de **modos de direccionamiento sencillos**. Casi todas las instrucciones RISC usan direccionamiento sencillo a registro. Se pueden incluir varios modos adicionales, como el desplazamiento y el relativo al contador de programa. Otros modos más complejos se pueden sintetizar por software a partir de los simples. Nuevamente, esta característica de diseño simplifica el repertorio de instrucciones y la unidad de control.

Una última característica común es el uso de **formatos de instrucción sencillos**. Generalmente, solo se usa un formato o unos pocos. La longitud de las instrucciones es fija y alineada en los límites

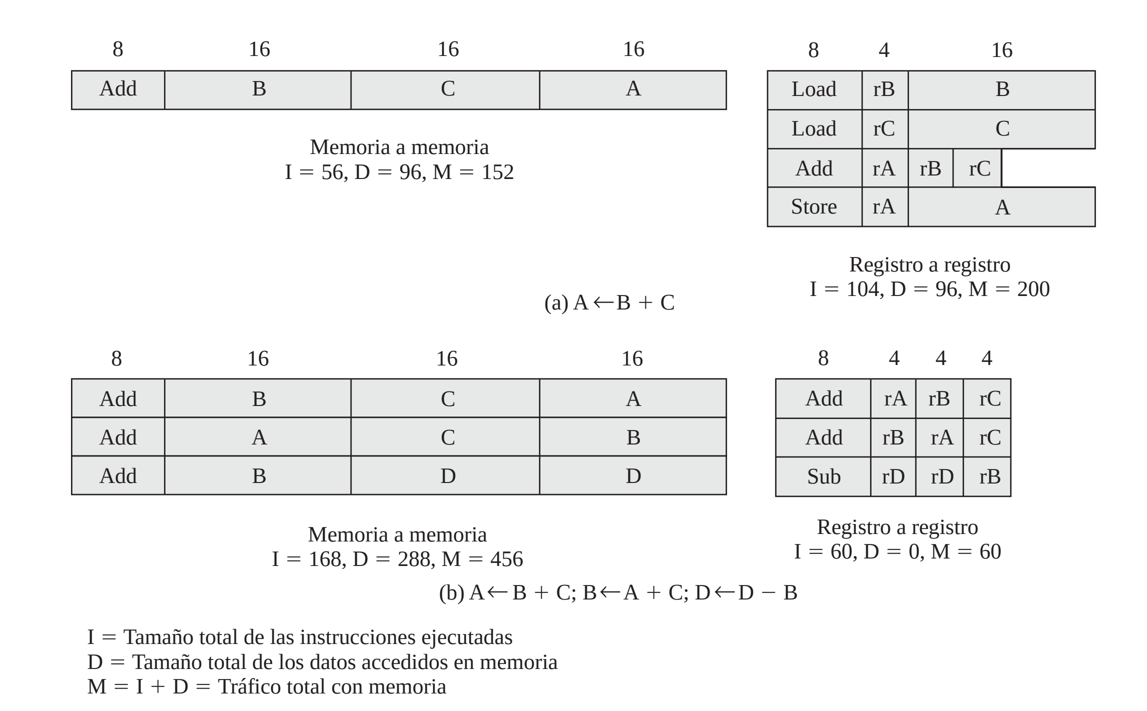


Figura 13.5. Dos comparaciones de las aproximaciones registro a registro y memoria a memoria.

de una palabra. Las posiciones de los campos, especialmente la del código de operación, son fijas. Este tipo de diseño tiene varias ventajas. Con campos fijos, la decodificación del código de operación y el acceso a los operandos en registros puede tener lugar simultáneamente. Los formatos sencillos simplifican la unidad de control. Se optimiza la captación de instrucciones ya que se captan unidades de una palabra de longitud. La alineación en el límite de una palabra también significa que una única instrucción no traspasa los límites de una página.

Estas características pueden evaluarse conjuntamente para determinar las ventajas potenciales de la aproximación RISC. Se puede presentar cierta cantidad de «pruebas indiciarias». En primer lugar, se pueden desarrollar compiladores con capacidad de optimización más eficaces. Con instrucciones más primitivas, hay más ocasiones de sacar funciones fuera de bucles, de reorganizar código buscando una mayor eficiencia, de maximizar la utilización de registros, etc. Es posible incluso calcular partes de instrucciones complejas en tiempo de compilación. Por ejemplo, la instrucción *Move Characters* (MVC) del S/390 transfiere una cadena de caracteres de una posición a otra. Cada vez que se ejecuta, la transferencia dependerá de la longitud de la cadena, de si las posiciones se solapan y en qué direcciones, y de cuáles son las características de alineación. En la mayoría de los casos, esto se conocerá en tiempo de compilación. Por tanto, el compilador podría producir una secuencia optimizada de instrucciones primitivas para esta función.

Un segundo punto, ya mencionado, es que la mayoría de las instrucciones generadas por un compilador son de todos modos relativamente sencillas. Parece razonable que una unidad de control construida expresamente para estas instrucciones y que usara poco o ningún microcódigo podría ejecutarlas más rápido que un CISC comparable.

Un tercer punto tiene que ver con el uso de segmentación de las instrucciones. Los investigadores de RISC creen que la técnica de segmentación de las instrucciones se puede aplicar mucho más eficazmente a un repertorio reducido de instrucciones. Más adelante examinaremos este punto en detalle.

Un último punto, tal vez de menor importancia, es que los programas RISC deberían de ser más sensibles a las interrupciones ya que estas se comprueban entre operaciones bastante elementales. Las arquitecturas con instrucciones complejas o bien restringen las interrupciones a los límites de instrucción o bien tienen que definir puntos interrumpibles específicos e implementar mecanismos para reanudar la ejecución de una instrucción.

La defensa de unas prestaciones mejoradas con una arquitectura de repertorio reducido de instrucciones es sólida, pero quizá se podría hacer aún un alegato a favor de CISC. Se han hecho varios estudios pero no con máquinas de tecnología y potencia comparables. Además, la mayoría de los estudios no han intentado separar los efectos de un repertorio reducido de instrucciones de los de un banco de registros extenso. La «prueba indiciaria», sin embargo, es sugerente.

CARACTERÍSTICAS CISC FRENTE A RISC

Tras el entusiasmo inicial por las máquinas RISC, ha habido una creciente convicción de que (1) los diseños RISC pueden sacar provecho de la inclusión de algunas características CISC y de que (2) los diseños CISC pueden sacar provecho de la inclusión de algunas características RISC. El resultado es que los diseños RISC más recientes, especialmente el PowerPC, no son ya RISC «puros» y que los diseños CISC más recientes, particularmente el Pentium II y los modelos de Pentium posteriores, incorporan ciertas características RISC.

Una comparación interesante en [MASH94] se adentra en este asunto. La Tabla 13.7 lista varios procesadores y los compara según diversas características. Para el objetivo de esta comparación, se considera típico de un RISC clásico lo siguiente:

1. Un único tamaño de instrucción.

2. Ese tamaño es típicamente cuatro bytes.

3. Un pequeño número de modos de direccionamiento de datos, típicamente menor que cinco. Este parámetro es difícil de precisar. En la tabla, los modos registro y literal no se han contado y los distintos formatos con diferentes tamaños de desplazamiento se han contado por separado.

4. No se usa direccionamiento indirecto que requiera efectuar un acceso a memoria para conseguir la dirección de memoria de otro operando.

5. No hay operaciones que combinen carga/almacenamiento con cálculos aritméticos (por ejemplo, suma desde memoria, suma a memoria).

6. No se direcciona más de un operando de memoria por instrucción.

7. Las operaciones de carga/almacenamiento no admiten una alineación de datos arbitraria.

8. Un número máximo de usos de la unidad de gestión de memoria (*Memory Management Unit*, MMU) de una dirección de dato en cada instrucción.

Tabla 13.7. Características de algunos procesadores.

Procesador	Número de tamaños de instrucción diferentes	Tamaño de instrucción máximo en bytes	Número de modos de direccionamiento	Direccionamiento indirecto	Carga/almacenamiento con cálculo aritmético asociado	Número máximo de operandos en memoria	Direccionamiento no alineado permitido	Número máximo de usos de la MMU	Número de bits por especificador de registro entero	Número de bits por especificador de registro de coma flotante
AMD29000	1	4	1	no	no	1	no	1	8	3
MIPS R2000	1	4	1	no	no	1	no	1	5	4
SPARC	1	4	2	no	no	1	no	1	5	4
MC88000	1	4	3	no	no	1	no	1	5	4
HP PA	1	4	10[a]	no	no	1	no	1	5	4
IBM RT/PC	2[a]	4	1	no	no	1	no	1	4[a]	3[a]
IBM RS/6000	1	4	4	no	no	1	sí[a]	1	5	5
Intel i860	1	4	4	no	no	1	no	1	5	4
IBM 3090	4	8	2[b]	no[b]	sí	2	sí	4	4	2
Intel 80486	12	12	15	no[b]	sí	2	sí	4	3	3
NSC 32016	21	21	23	sí	sí	2	sí	4	3	3
MC68040	11	22	44	sí	sí	2	sí	8	4	3
VAX	56	56	22	sí	sí	6	sí	24	4	0
Clipper	4[a]	8[a]	9[a]	no	no	1	no	2	4[a]	3[a]
Intel 80960	2[a]	8[a]	9[a]	no	no	1	sí[a]	—	5	3[a]

[a] RISC que no se ajusta a esta característica.
[b] CISC que no se ajusta a esta característica.

9. El número de bits de un campo designador de registro entero es de cinco o más. Esto quiere decir que, en un momento dado, se pueden referenciar explícitamente por lo menos 32 registros enteros.
10. El número de bits de un campo designador de registro de coma flotante es de cuatro o más. Esto quiere decir que por lo menos 16 registros de coma flotante se pueden referenciar explícitamente en un momento dado.

Los puntos 1 a 3 indican la complejidad de la decodificación de instrucciones. Los puntos 4 a 8 indican la facilidad o dificultad de la segmentación, especialmente por la presencia de requisitos de memoria virtual. Los puntos 9 y 10 se relacionan con la habilidad de sacar partido de los compiladores.

En la tabla, los primeros ocho procesadores poseen claramente arquitectura RISC, los siguientes cinco son sin duda CISC, y los últimos dos procesadores son considerados a menudo como RISC que en realidad tienen muchas características CISC.

13.5. SEGMENTACIÓN EN RISC

SEGMENTACIÓN CON INSTRUCCIONES REGULARES

Como discutimos en la Sección 12.4, para aumentar las prestaciones se usa con frecuencia la segmentación de instrucciones. Reconsideremos este asunto en el contexto de la arquitectura RISC. La mayoría de las instrucciones son del tipo registro a registro, y un ciclo de instrucción tiene las dos fases siguientes:

- I: captación de instrucción.
- E: ejecución. Realiza una operación de la ALU con registros como entrada y salida.

Las operaciones de carga y almacenamiento necesitan tres fases:

- I: captación de instrucción.
- E: ejecución. Calcula una dirección de memoria.
- D: memoria. Operación registro a memoria o memoria a registro.

La Figura 13.6a representa la temporización de una secuencia de instrucciones que no usan segmentación. Sin duda, se trata de un proceso costoso. Incluso una segmentación muy simple puede mejorar las prestaciones sustancialmente. La Figura 13.6b muestra un esquema de segmentación de dos etapas, en el cual las etapas I y E de dos instrucciones diferentes se pueden ejecutar simultáneamente. Este esquema ofrece el doble de velocidad de ejecución que un esquema serie. Hay dos problemas que impiden que se consiga la máxima aceleración. El primero es que suponemos que se usa una memoria de un único puerto y que solo es posible un acceso a memoria en cada etapa. Esto requiere la inserción de un estado de espera en algunas instrucciones. El segundo problema consiste en que una instrucción de salto interrumpe el flujo secuencial de ejecución. Para adaptarse a estos problemas con la mínima circuitería, el compilador o ensamblador puede insertar una instrucción NOOP en el flujo de instrucciones.

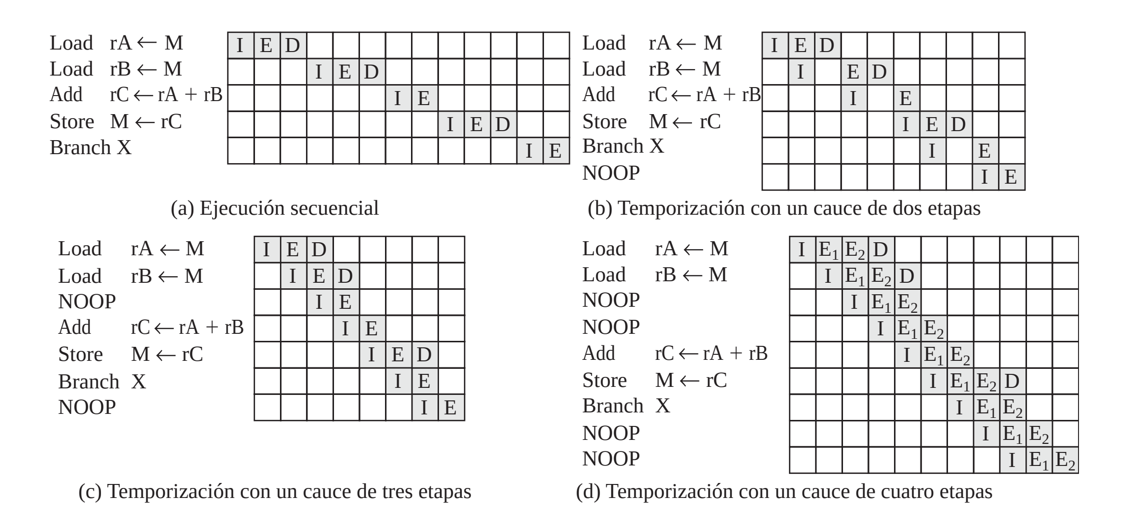


Figura 13.6. Efectos de la segmentación.

La segmentación puede ser mejorada permitiendo dos accesos a memoria por etapa. Esto produce la secuencia que se muestra en la Figura 13.6c. Ahora, hasta tres instrucciones pueden solaparse, y la mejora es como mucho un factor de 3. Nuevamente, las instrucciones de salto hacen que la aceleración sea un poco menor que la máxima posible. Hay que advertir también que las dependencias de datos tienen su efecto. Si una instrucción necesita un operando que es modificado por la instrucción precedente, se necesita un retardo. También esto puede ser llevado a cabo por un NOOP.

La segmentación discutida hasta aquí funciona mejor si las tres etapas son aproximadamente de la misma duración. Como la etapa E suele implicar una operación de la ALU, puede ser más larga. En ese caso, podemos dividirla en dos subetapas:

- E_1: lectura del banco de registros.
- E_2: operación de la ALU y escritura en el registro.

Dada la simplicidad y regularidad de un repertorio de instrucciones RISC, el diseño de la organización en tres o cuatro etapas se realiza fácilmente. La Figura 13.6d muestra el resultado con un cauce de cuatro etapas. Hasta cuatro instrucciones pueden estar en curso en un momento dado, y la aceleración potencial máxima es un factor de 4. Observe que de nuevo hay que usar NOOP para los retardos por saltos y dependencias de datos.

OPTIMIZACIÓN DE LA SEGMENTACIÓN

Dada la naturaleza sencilla y regular de las instrucciones RISC, los esquemas de segmentación se pueden emplear eficientemente. Hay poca variación en la duración de la ejecución de instrucciones, y el cauce puede adaptarse para reflejar este hecho. Sin embargo, hemos visto que las dependencias de datos y saltos reducen la velocidad de ejecución global.

Para compensar estas dependencias, se han desarrollado técnicas de reorganización del código. Consideremos primero las instrucciones de salto. El *salto retardado*, que es una forma de incrementar la eficiencia de la segmentación, utiliza un salto que no tiene lugar hasta después de que se ejecute la siguiente instrucción (de ahí el término *retardado*). La posición de la instrucción situada inmediatamente después de la instrucción de salto se conoce como *espacio de retardo*. Este curioso procedimiento se ilustra en la Tabla 13.8. En la columna «salto normal», vemos un programa normal

Tabla 13.8. Salto normal y salto retardado.

Dirección	Salto normal		Salto retardado		Salto retardado optimizado	
100	LOAD	X, rA	LOAD	X, rA	LOAD	X, rA
101	ADD	1, rA	ADD	1, rA	JUMP	105
102	JUMP	105	JUMP	106	ADD	1, rA
103	ADD	rA, rB	NOOP		ADD	rA, rB
104	SUB	rC, rB	ADD	rA, rB	SUB	rC, rB
105	STORE	rA, Z	SUB	rC, rB	STORE	rA, Z
106			STORE	rA, Z		

en lenguaje máquina con instrucciones simbólicas. Después de que se ejecute la instrucción de la dirección 102, la siguiente instrucción a ejecutar es la de la dirección 105. Para regularizar el cauce, se inserta un NOOP después de este salto. No obstante, se pueden incrementar las prestaciones si se intercambian las instrucciones 101 y 102.

La Figura 13.7 muestra el resultado. La Figura 13.7a muestra la solución tradicional a la segmentación, del tipo discutido en el Capítulo 12 (por ejemplo, *ver* Figuras 12.11 y 12.12). La instrucción JUMP se capta en el instante 4. En el instante 5, la instrucción JUMP se ejecuta a la vez que se capta la instrucción 103 (instrucción ADD). Como se ejecuta una instrucción JUMP, que actualiza el contador de programa, la instrucción 103 tiene que ser eliminada del cauce; en el instante 6, se carga la instrucción 105, que es el destino de la instrucción JUMP. La Figura 13.7b muestra el mismo cauce gestionado por una organización RISC típica. La temporización es parecida. Sin embargo, debido a la inserción de la instrucción NOOP, no es necesaria ninguna circuitería especial para vaciar el cauce; el NOOP sencillamente se ejecuta sin ningún efecto. La Figura 13.7c muestra el uso del salto retardado. La instrucción JUMP se capta en el instante 2, antes que la instrucción ADD, que se capta en el instante 3. Hay que observar, sin embargo, que la instrucción ADD se capta antes de que la ejecución de la instrucción JUMP tenga la oportunidad de modificar el contador de programa. Por consiguiente, durante el instante 4, la instrucción ADD se ejecuta a la vez que se capta la instrucción 105. De este modo, se conserva la semántica original del programa pero se necesita un ciclo de reloj menos para la ejecución.

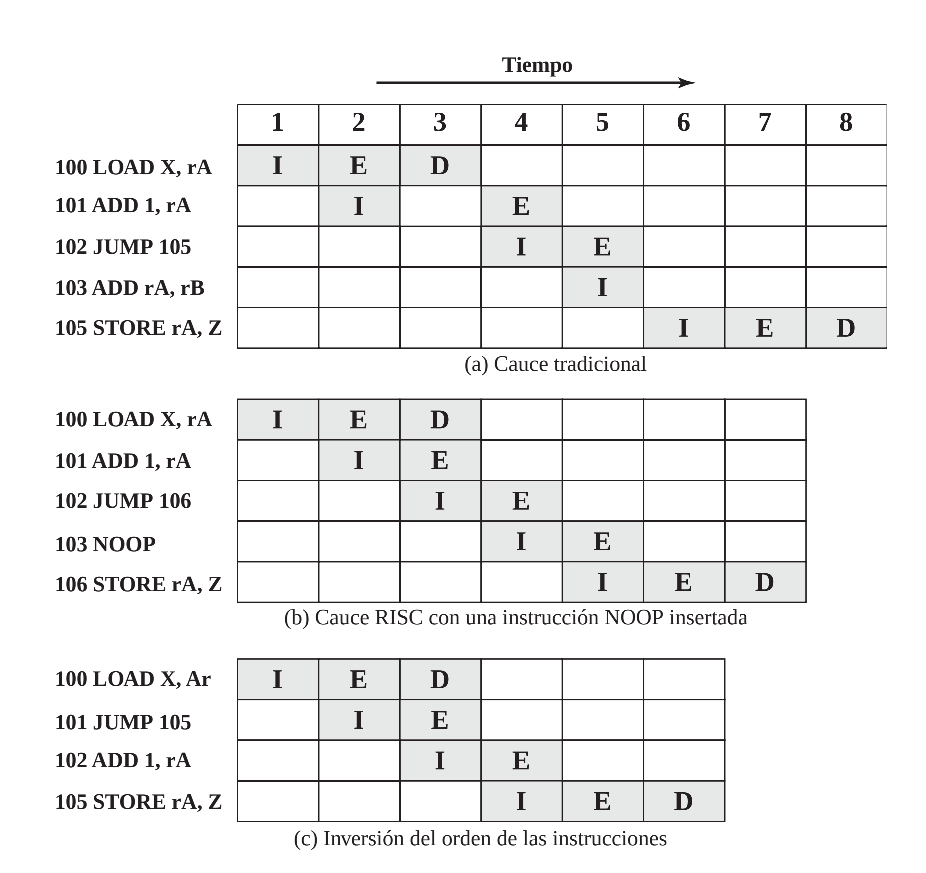


Figura 13.7. Uso del salto retardado.

Este intercambio de instrucciones funcionará con éxito con saltos incondicionales, llamadas y retornos. Para saltos condicionales, el procedimiento no se puede aplicar a ciegas. Si la condición comprobada por la bifurcación puede alterarse por la instrucción inmediatamente precedente, el compilador ha de abstenerse de hacer el intercambio y en su lugar debe insertar un NOOP. En caso contrario, el compilador puede intentar insertar una instrucción útil después del salto. La experiencia con los sistemas RISC de Berkeley e IBM 801 es que la mayoría de instrucciones de salto condicional pueden optimizarse de esta forma ([PATT82a], [RADI83]).

Un tipo de táctica similar, llamada carga retardada, se puede usar con las instrucciones LOAD. En las instrucciones LOAD, el procesador bloquea el registro destino de la carga. Después el procesador continúa la ejecución del flujo de instrucciones hasta que alcanza una instrucción que necesite ese registro, deteniéndose hasta que la carga finalice. Si el compilador puede reorganizar las instrucciones de manera que se pueda hacer un trabajo útil mientras la carga está en el cauce, la eficiencia aumenta.

Como nota final, hay que señalar que el diseño del cauce de instrucciones no debe ser llevado a cabo independientemente de otras técnicas de optimización aplicadas al sistema. Por ejemplo, [BRAD91b] muestra que la planificación de instrucciones para el cauce y la asignación dinámica de registros tienen que considerarse conjuntamente para lograr la mayor eficiencia.

13.6. MIPS R4000

Uno de los primeros conjuntos de chips RISC disponibles comercialmente fue desarrollado por MIPS Technology Inc. El sistema se inspiró en un sistema experimental, que también usaba el nombre MIPS, desarrollado en Stanford [HENN84]. En esta sección estudiamos el MIPS R4000. Tiene sustancialmente la misma arquitectura y repertorio de instrucciones de los anteriores diseños MIPS: el R2000 y el R3000. La diferencia más significativa es que el R4000 usa 64 en lugar de 32 bits para los buses de datos internos, externos, para las direcciones, los registros y la ALU.

El uso de 64 bits tiene diversas ventajas sobre una arquitectura de 32 bits. Permite un espacio de direccionamiento más grande —suficientemente grande para que un sistema operativo coloque más de un terabyte de ficheros directamente en la memoria virtual para acceder fácilmente. Siendo corrientes en este momento discos de un gigabyte y mayores, el espacio de direcciones de cuatro gigabytes de una máquina de 32 bits se vuelve limitado. También, la capacidad de 64 bits permite al R4000 procesar datos tales como números en coma flotante IEEE de doble precisión y cadenas de hasta ocho caracteres de una sola vez.

El chip del procesador R4000 está dividido en dos secciones, una contiene la CPU, y la otra contiene un coprocesador de gestión de memoria. El procesador tiene una arquitectura muy sencilla. El propósito fue diseñar un sistema en el cual la lógica de ejecución de instrucciones fuera lo más sencilla posible, dejando espacio disponible para la lógica que aumentara las prestaciones (por ejemplo, la unidad de gestión de memoria completa).

El procesador contiene 32 registros de 64 bits. También está provisto de hasta 128 Kbytes de caché de alta velocidad, la mitad para instrucciones y la mitad para datos. Esta caché relativamente grande (el IBM 3090 está provisto de 128 a 256 Kbytes de caché) permite que el sistema mantenga grandes conjuntos de código de programa y datos locales al procesador, descargando el bus de memoria principal y evitando la necesidad de un banco de registros grande con la lógica de ventanas asociada.

REPERTORIO DE INSTRUCCIONES

La Tabla 13.9 lista el repertorio de instrucciones básico de todos los procesadores de la serie MIPS R. Todas las instrucciones del procesador se codifican en un único formato de palabra de 32 bits. Todas

Tabla 13.9. Repertorio de instrucciones de la serie R de MIPS.

OP	Descripción
	Instrucciones de carga/almacenamiento
LB	Cargar byte
LBU	Cargar byte sin signo
LH	Cargar media palabra
LHU	Cargar media palabra sin signo
LW	Cargar palabra
LWL	Cargar palabra a la izquierda
LWR	Cargar palabra a la derecha
SB	Almacenar byte
SH	Almacenar media palabra
SW	Almacenar palabra
SWL	Almacenar palabra a la izquierda
SWL	Almacenar palabra a la derecha
	Instrucciones aritméticas (inmediatas con la ALU)
ADDI	Sumar inmediato
ADDIU	Sumar inmediato sin signo
SLTI	Poner a uno si menor inmediato
SLTIU	Poner a uno si menor inmediato sin signo
ANDIY	Inmediato
ORIO	Inmediato
XORIO	Exclusivo inmediato
LUI	Cargar superior inmediato
	Instrucciones aritméticas (tres operandos, tipo registro)
ADD	Sumar
ADDU	Sumar sin signo
SUB	Restar
SUBU	Restar sin signo
SLT	Poner a uno si menor
SLTU	Poner a uno si menor sin signo
AND	Y
OR	O
XORO	Exclusiva
NOR	No O
	Instrucciones de desplazamiento
SLL	Desplazamiento lógico a la izquierda
SRL	Desplazamiento lógico a la derecha
SRA	Desplazamiento aritmético a la derecha
SLLV	Desplazamiento lógico a la izquierda variable
SRLV	Desplazamiento lógico a la derecha variable
SRAV	Desplazamiento aritmético a la derecha variable
	Instrucciones de multiplicación/división
MULT	Multiplicar
MULTU	Multiplicar sin signo
DIV	Dividir
DIVU	Dividir sin signo
MFHI	Transferir desde parte alta
MTHI	Transferir a parte alta
MFLO	Transferir desde parte baja
MTLO	Transferir a parte baja
	Instrucciones de salto y bifurcación
J	Saltar
JAL	Saltar y enlazar
JR	Saltar a registro
JALR	Saltar y enlazar registro
BEQ	Bifurcar si igual
BNE	Bifurcar si distinto
BLEZ	Bifurcar si menor o igual que cero
BGTZ	Bifurcar si mayor que cero
BLTZ	Bifurcar si menor que cero
BGEZ	Bifurcar si mayor o igual que cero
BLTZAL	Bifurcar si menor que cero y enlazar
BGEZAL	Bifurcar si mayor o igual que cero y enlazar
	Instrucciones del coprocesador
LWCz	Cargar palabra en el coprocesador
SWCz	Almacenar palabra en el coprocesador
MTCz	Transferir al coprocesador
MFCz	Transferir desde el coprocesador
CTCz	Transferir control al coprocesador
CFCz	Transferir control desde el coprocesador
COPz	Operación del coprocesador
BCzT	Bifurcar si coprocesador z es cierto
BCzF	Bifurcar si coprocesador z es falso
	Instrucciones especiales
SYSCALL	Llamada al sistema
BREAK	Ruptura

las operaciones con datos son del tipo registro a registro; las únicas referencias a memoria son operaciones puras de carga/almacenamiento.

El R4000 no utiliza códigos de condición. Si una instrucción genera una condición, los indicadores correspondientes se almacenan en un registro de uso general. Esto elimina la necesidad de una lógica especial para tratar con códigos de condición ya que estos afectan al mecanismo de segmentación y a la reordenación de instrucciones por el compilador. En lugar de eso, se emplea el mecanismo ya implementado que se ocupa de las dependencias de valores de registros. Además, las condiciones asignadas en el banco de registros son propensas a las mismas optimizaciones del compilador en cuanto a asignación y reutilización que cualquier otro valor almacenado en un registro.

Como la mayoría de las máquinas de tipo RISC, el MIPS usa un tamaño de instrucción fijo de 32 bits. Esta única longitud de instrucción simplifica la captación y la decodificación de instrucciones, y también simplifica la interacción entre la captación de instrucciones y la unidad de gestión de memoria virtual (esto es, las instrucciones no atraviesan los límites de una palabra o una página). Los tres formatos de instrucción (Figura 13.8) comparten un formato común para los códigos de operación y las referencias a registros, lo que simplifica la decodificación de instrucciones. La acción de instrucciones más complejas se puede sintetizar en tiempo de compilación.

Sólo se implementa en hardware el modo de direccionamiento a memoria que se usa con más frecuencia. Todas las referencias a memoria constan de un desplazamiento de 16 bits y de un registro de 32 bits. Por ejemplo, la instrucción «cargar palabra» («*load word*») tiene la forma

lw r2, 128(r3)	carga la palabra de la dirección indicada en el registro 3 más un desplazamiento de 128 en el registro 2

Cualquiera de los 32 registros de uso general puede usarse como registro base. Un registro, el r0, siempre contiene 0.

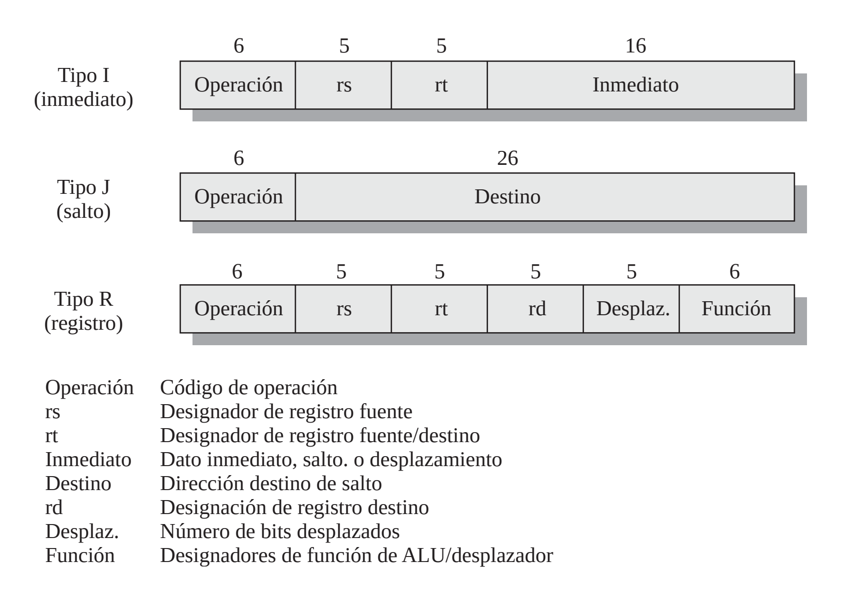


Figura 13.8. Formatos de instrucciones del MIPS.

Tabla 13.10. Síntesis de otros modos de direccionamiento a partir de los modos de direccionamiento del MIPS.

Instrucción aparente	Instrucción real
lw r2, <desplazamiento de 16 bits>	lw r2, <desplazamiento de 16 bits> (r0)
lw r2, <desplazamiento de 32 bits>	lui r1, <16 bits altos del desplazamiento> lw r2, <16 bits bajos del desplazamiento> (r1)
lw r2, <desplazamiento de 32 bits> (r4)	lui r1, <16 bits altos del desplazamiento> addu r1, r1, r4 lw r2, <16 bits bajos del desplazamiento> (r1)

El compilador usa múltiples instrucciones máquina para sintetizar los modos de direccionamiento de las máquinas convencionales. Se dan algunos ejemplos en la Tabla 13.10 [CHOW87]. La tabla muestra el uso de la instrucción LUI (*Load Upper Immediate*, cargar superior inmediato). La instrucción carga la mitad superior de un registro con un valor inmediato de 16 bits, poniendo la mitad inferior a cero.

CAUCE DE INSTRUCCIONES

Gracias a su arquitectura de instrucciones simplificada, el MIPS puede lograr una segmentación muy eficiente. Es instructivo estudiar la evolución del cauce del MIPS, ya que ilustra la evolución de la segmentación en los RISC en general.

Los sistemas RISC experimentales iniciales y la primera generación de los procesadores RISC comerciales lograban velocidades de ejecución que se aproximaban a una instrucción por ciclo de reloj del sistema. Para mejorar estas prestaciones, han evolucionado dos clases de procesadores que ejecutan múltiples instrucciones por ciclo de reloj: las arquitecturas superescalares y las supersegmentadas. Esencialmente, una arquitectura superescalar reproduce exactamente cada etapa del cauce de manera que dos o más instrucciones en la misma etapa del cauce se puedan procesar simultáneamente. Una arquitectura supersegmentada es aquella que utiliza más etapas, y de grano más fino, en el cauce. Con más etapas puede haber más instrucciones en el cauce al mismo tiempo, aumentando el paralelismo.

Las dos alternativas tienen limitaciones. En la segmentación superescalar, las dependencias entre instrucciones de cauces diferentes pueden reducir la velocidad del sistema. Además se necesita lógica adicional para coordinar estas dependencias. En la supersegmentación, hay ciertos gastos extra asociados con la transferencia de instrucciones de una etapa a la siguiente.

El Capítulo 14 se dedica al estudio de la arquitectura superescalar. El MIPS R4000 es un buen ejemplo de arquitectura RISC supersegmentada.

La Figura 13.9a muestra el cauce de instrucciones del R3000. En el R3000, el cauce avanza una vez por ciclo de reloj. El compilador del MIPS es capaz de reordenar las instrucciones para rellenar con código los espacios de retardo el setenta a noventa por ciento de las veces. Todas las instrucciones siguen la misma secuencia de cinco etapas del cauce:

- Captación de instrucción.

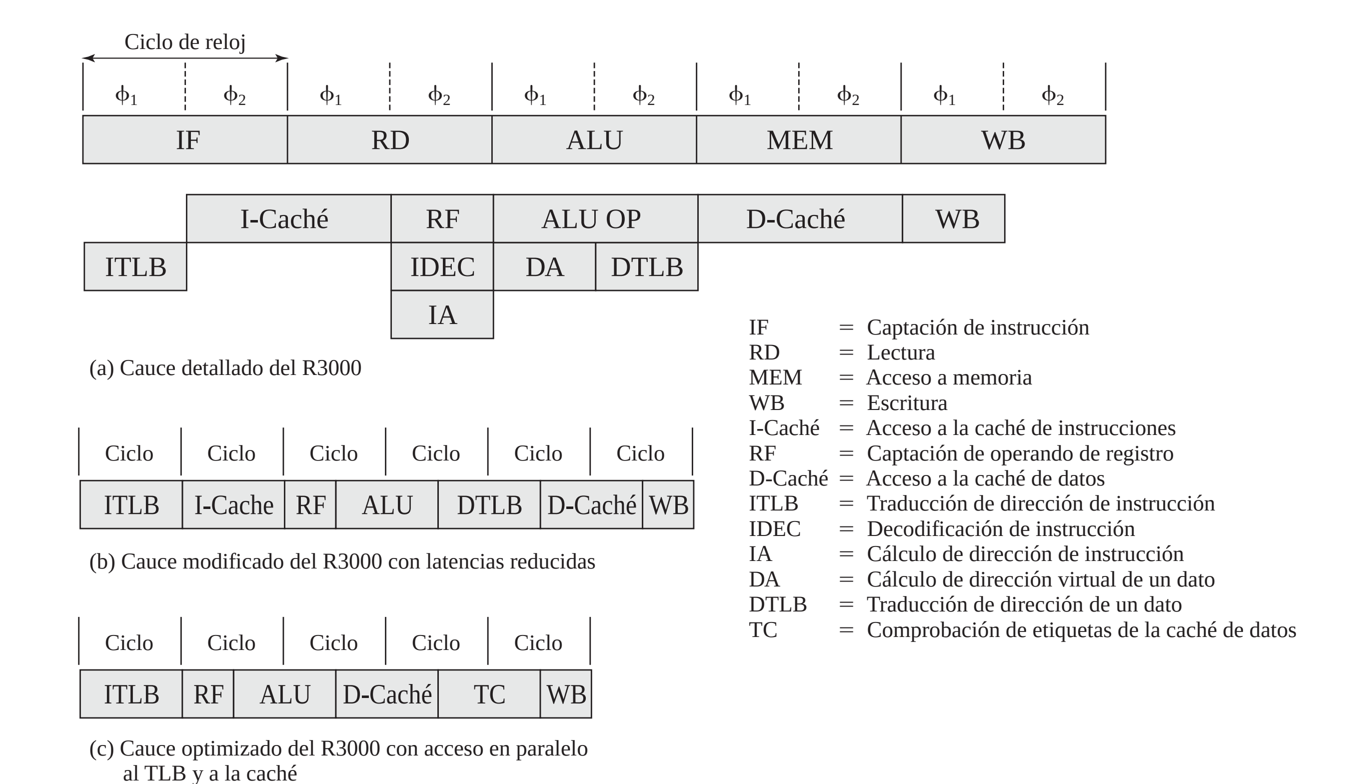


Figura 13.9. Mejoras del cauce del R3000.

- Captación de operando fuente del banco de registros.
- Operación en la ALU o generación de dirección de operando.
- Referencia a dato en memoria.
- Escritura en el banco de registros.

Como ilustra la Figura 13.9a, no solo hay paralelismo debido a la segmentación sino también paralelismo en la ejecución de una única instrucción. El ciclo de reloj de 60 ns se divide en dos fases de 30 ns. Las operaciones externas de acceso a instrucciones y datos de la caché requieren 60 ns cada una, igual que las principales operaciones internas (OP, DA, IA). La decodificación de instrucciones es una operación más sencilla, y requiere solo una fase de 30 ns, solapada con la captación de registros de la misma instrucción. El cálculo de una dirección en una instrucción de salto también se superpone a la decodificación de la instrucción y a la captación de registros, de modo que un salto en la instrucción *i* pueda direccionar el acceso I-Caché de la instrucción $i + 2$. De un modo parecido, una carga en la instrucción *i* capta datos usados inmediatamente por la fase OP de la instrucción $i + 1$, mientras que un resultado de la ALU o de un desplazamiento se pasa directamente a la instrucción $i + 1$ sin retardo alguno. Este estrecho acoplamiento entre las instrucciones conduce a un cauce muy eficiente.

Detalladamente, por tanto, cada ciclo de reloj se divide en fases separadas, que llamamos $\phi 1$ y $\phi 2$. Las funciones realizadas en cada fase se resumen en la Tabla 13.11.

Tabla 13.11. Etapas del cauce del R3000.

Etapa del cauce	Fase	Función
IF	$\phi 1$	Usando el TLB, traducir la dirección virtual de una instrucción a dirección física (después de una decisión de salto).
IF	$\phi 2$	Enviar la dirección física a la caché de instrucciones.
RD	$\phi 1$	Devolver una instrucción desde la caché de instrucciones. Comparar etiquetas y validez de la instrucción captada.
RD	$\phi 2$	Decodificar la instrucción. Leer del banco de registros. Si hay un salto, calcular la dirección destino.
ALU	$\phi 1 + \phi 2$	Si se trata de una operación registro a registro, se ejecuta la operación aritmética o lógica.
ALU	$\phi 1$	Si se trata de un salto, decidir si se produce o no. Si se trata de una referencia a memoria (carga o almacenamiento), calcular la dirección virtual del dato.
ALU	$\phi 2$	Si se trata de una referencia a memoria, traducir la dirección virtual del dato a dirección física usando el TLB.
MEM	$\phi 1$	Si se trata de una referencia a memoria, enviar la dirección física a la caché de datos.
MEM	$\phi 2$	Si se trata de una referencia a memoria, devolver el dato desde la caché de datos, y comprobar las etiquetas.
WB	$\phi 1$	Escribir en el banco de registros.

El R4000 incorpora varios avances técnicos sobre el R3000. La utilización de una tecnología más avanzada posibilita que se reduzca a la mitad, a 30 ns, el periodo de reloj, y que el tiempo de acceso al banco de registros también se reduzca a la mitad. Además, la mayor densidad del circuito permite que las cachés de instrucciones y de datos se integren en el mismo chip. Antes de examinar el cauce final del R4000, consideremos como se puede modificar el cauce del R3000 para mejorar las prestaciones usando la tecnología del R4000.

La Figura 13.9b muestra un primer paso. Recuerde que los ciclos de esta figura duran la mitad que los de la Figura 13.9a. Debido a que están en el mismo chip, las etapas de acceso a las cachés de instrucciones y de datos suponen solo la mitad de tiempo; de tal modo que aún duran un único ciclo de reloj. Además, debido al aumento de velocidad en el acceso al banco de registros, la lectura y escritura de un registro sigue durando solo la mitad de un ciclo de reloj.

Como las cachés del R4000 están integradas en el chip, la traducción de dirección virtual a física puede retrasar el acceso a caché. Este retardo se reduce implementando cachés indexadas virtualmente y yendo hacia una paralelización del acceso a la caché y de la traducción de la dirección. La Figura 13.9c muestra el cauce del R3000 optimizado con esta mejora. A causa de la compresión de eventos, la comprobación de etiquetas de la caché de datos se realiza por separado en el ciclo posterior al de acceso a la caché.

En un sistema supersegmentado, el hardware existente se usa varias veces por ciclo insertando registros de separación para dividir cada etapa del cauce. Básicamente, cada etapa del cauce supersegmentado funciona en un múltiplo de la frecuencia de reloj base, dependiendo el múltiplo del grado de supersegmentación. La tecnología del R4000 tiene una velocidad y una densidad que permite una supersegmentación de grado 2. La Figura 13.10a muestra el cauce del R3000 optimizado usando esta supersegmentación. Observe que se trata esencialmente de la misma estructura dinámica de la Figura 13.9c.

Se pueden llevar a cabo más mejoras. Para el R4000, se diseñó un sumador mucho más grande y especializado. Esto posibilita la ejecución de operaciones de la ALU a doble velocidad. Otras mejoras permiten doblar la velocidad de ejecución de cargas y almacenamientos. El cauce resultante se muestra en la Figura 13.10b.

El R4000 tiene un cauce de ocho etapas, lo que quiere decir que puede haber hasta ocho instrucciones en el cauce al mismo tiempo. El cauce avanza a un ritmo de dos etapas por ciclo de reloj. Las ocho etapas del cauce son:

- **Primera mitad de la captación de instrucción:** la dirección virtual se da a la caché de instrucciones y al buffer de traducción anticipada.
- **Segunda mitad de la captación de instrucción:** se obtiene la instrucción de la caché de instrucciones y el TLB genera la dirección física.
- **Banco de registros:** ocurren tres actividades en paralelo:
  - La instrucción se decodifica y se comprueban condiciones de interbloqueo (es decir, si esta instrucción depende del resultado de la instrucción precedente).
  - Se comprueba la etiqueta en la caché de instrucciones.
  - Se captan los operandos del banco de registros.

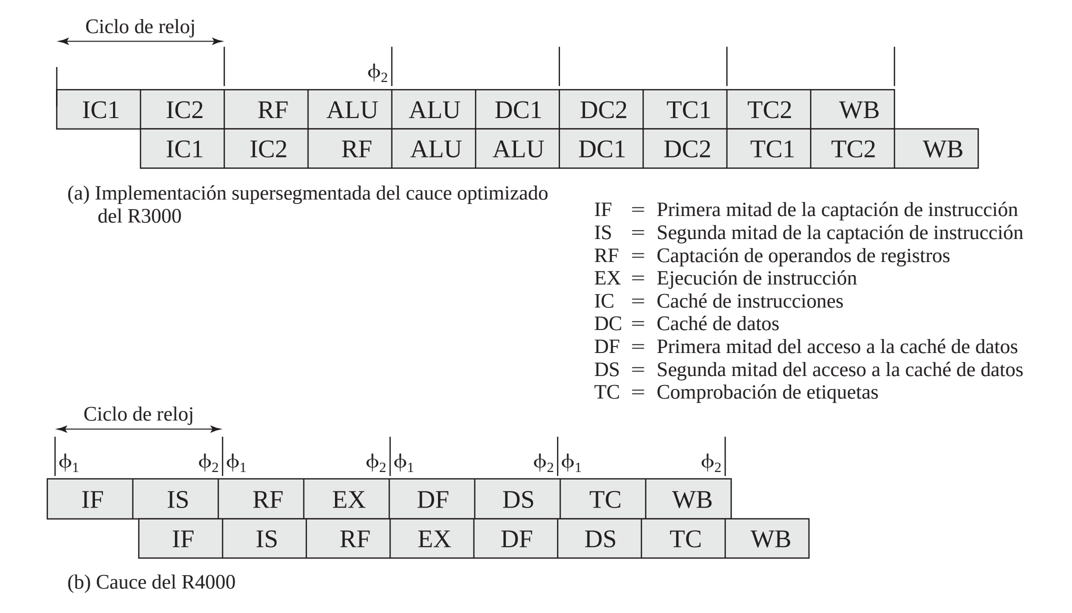


Figura 13.10. Cauces supersegmentados: teórico en el R3000 y real en el R4000.

- **Ejecución de instrucción:** puede ocurrir una de estas tres actividades:
  - Si la instrucción es una operación registro a registro, la ALU lleva a cabo la operación aritmética o lógica.
  - Si la instrucción es una carga o un almacenamiento, se calcula la dirección virtual del dato.
  - Si la instrucción es un salto, se calcula la dirección virtual del destino del salto y se comprueban las condiciones de salto.
- **Primera mitad del acceso a la caché de datos:** la dirección virtual se da a la caché de datos y al TLB.
- **Segunda mitad del acceso a la caché de datos:** se obtiene el dato de la caché de datos, y el TLB genera la dirección física.
- **Comprobación de etiquetas:** se comprueban las etiquetas en la caché en el caso de cargas y almacenamientos.
- **Escritura:** el resultado de la instrucción se escribe en el banco de registros.

13.7. SPARC

El acrónimo SPARC (*Scalable Processor Architecture,* arquitectura de procesador escalable) hace referencia a una arquitectura definida por Sun Microsystems. Sun desarrolló su propia implementación de SPARC, pero también autoriza a otros vendedores a fabricar máquinas compatibles con SPARC. La arquitectura SPARC se inspira en la máquina RISC I de Berkeley, y su repertorio de instrucciones y organización de registros están basados exactamente en el modelo RISC de Berkeley.

CONJUNTO DE REGISTROS DEL SPARC

Como el RISC de Berkeley, el SPARC utiliza ventanas de registros. Cada ventana consta de 24 registros, y el número total de ventanas depende de la implementación y varía de 2 a 32 ventanas. La Figura 13.11 ilustra una implementación que admite ocho ventanas, con un total de 136 registros físicos; como indica la discusión de la Sección 13.2, este parece un número razonable de ventanas. Los registros físicos del 0 al 7 son registros globales compartidos por todos los procedimientos. Cada proceso ve los registros lógicos del 0 al 31. Los registros lógicos del 24 al 31, a los que se denomina *entradas*, son compartidos con el procedimiento que hace la llamada (padre); y los registros lógicos del 8 al 15, llamados *salidas*, son compartidos con cualquier procedimiento llamado (hijo). Estas dos partes se solapan con otras ventanas. Los registros lógicos del 16 al 23, denominados *locales*, no se comparten ni se superponen con otras ventanas. Nuevamente, como indica la discusión de la Sección 12.1, la disponibilidad de ocho registros para pasar parámetros debería ser suficiente en la mayoría de los casos (*ver,* por ejemplo, la Tabla 13.4).

La Figura 13.12 es otra representación de la superposición de registros. El procedimiento que llama coloca los parámetros a pasar en los registros *salidas*; el procedimiento llamado trata estos mismos registros físicos como sus registros *entradas*. El procesador guarda un puntero de ventana en curso (*Current-Window Pointer,* CWP), en el registro de estado del procesador (*Processor Status*

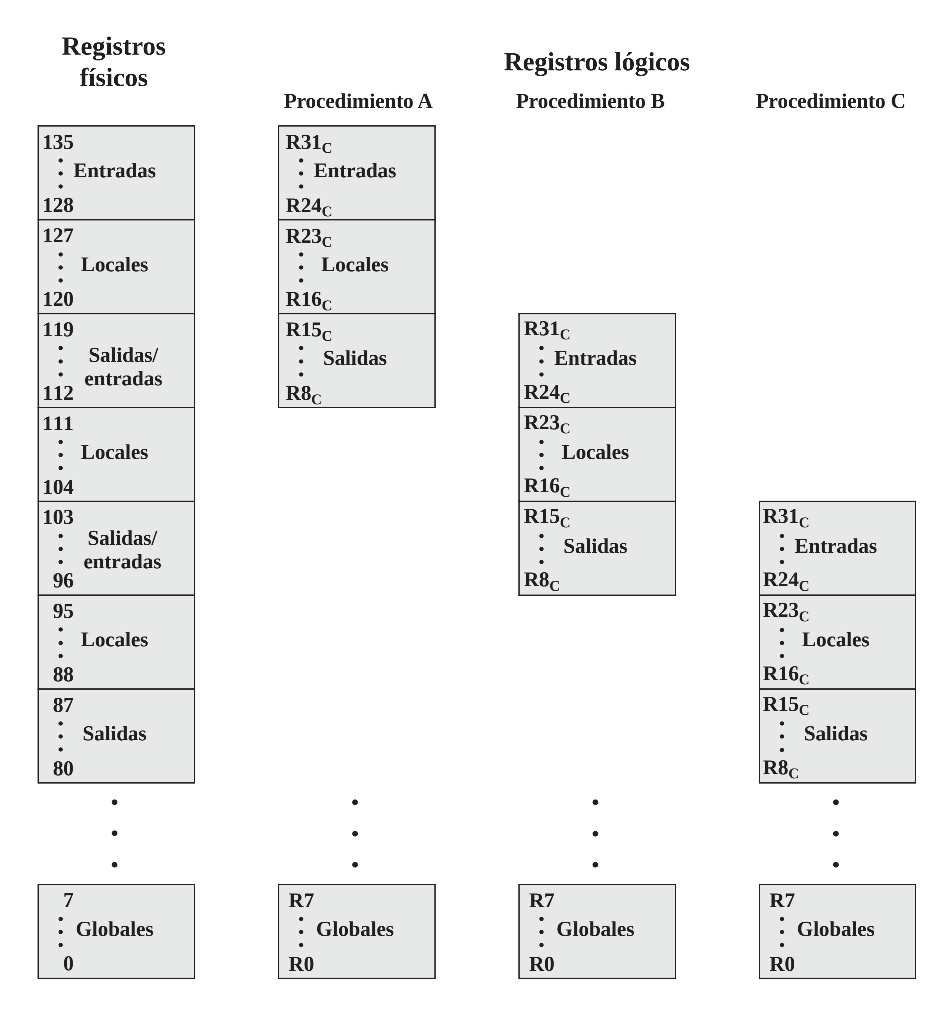


Figura 13.11. Disposición de las ventanas de registros del SPARC para tres procedimientos.

register, PSR), que apunta a la ventana del procedimiento en ejecución. La máscara de ventana no válida (*Window Invalid Mask*, WIM), también en el PSR, indica qué ventanas no son válidas.

Con la arquitectura de registros del SPARC, normalmente no es necesario guardar y restaurar ningún registro en una llamada a un procedimiento. El compilador se simplifica porque solo ha de preocuparse de la asignación eficiente de los registros locales de un procedimiento y no de la asignación de registros entre procedimientos.

REPERTORIO DE INSTRUCCIONES

La Tabla 13.12 lista las instrucciones de la arquitectura SPARC. La mayoría de las instrucciones referencian solamente operandos en registros. Las instrucciones registro a registro tienen tres operandos y se pueden expresar de la forma

$$R_d \leftarrow R_{S1} \text{ op } S2$$

R_d y R_{S1} son referencias a registros; S2 puede hacer referencia a un registro o a un operando inmediato de 13 bits. El registro cero (R_0) está cableado al valor 0. Este formato se adapta bien a los programas típicos, que tienen una alta proporción de datos escalares locales y constantes.

Las operaciones disponibles en la ALU se pueden agrupar como sigue:

- Suma entera (con o sin acarreo).
- Resta entera (con o sin adeudo).
- Operaciones booleanas bit a bit Y, O, O exclusiva y sus negaciones.
- Desplazamientos lógico a la izquierda, lógico a la derecha y aritmético a la derecha.

Todas estas instrucciones, excepto los desplazamientos, pueden ajustar opcionalmente los cuatro códigos de condición (CERO, NEGATIVO, DESBORDAMIENTO, ACARREO). Los enteros se representan con 32 bits en complemento a dos.

Únicamente las sencillas instrucciones de carga y almacenamiento referencian la memoria. Hay instrucciones de carga y almacenamiento separadas para palabras (32 bits), dobles palabras, medias palabras y bytes. Para los dos últimos casos, hay instrucciones que cargan estas cantidades como números con signo o sin signo. En el caso de números con signo se extiende el bit de signo para rellenar el registro destino de 32 bits. En el caso de números sin signo este se rellena con ceros.

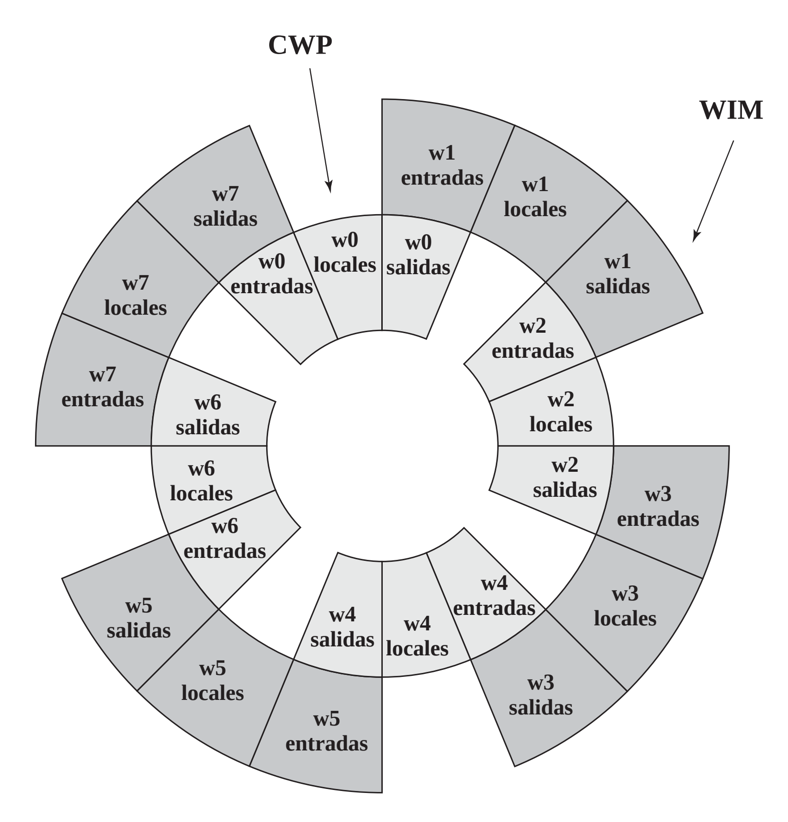


Figura 13.12. Ocho ventanas de registros que forman una pila circular en el SPARC.

Tabla 13.12. Repertorio de instrucciones del SPARC.

OP	Descripción	OP	Descripción
Instrucciones de carga/almacenamiento		**Instrucciones aritméticas**	
LDSB	Cargar byte con signo	ADD	Sumar
LDSH	Cargar media palabra con signo	ADDCC	Sumar, ajustar icc
LDUB	Cargar byte sin signo	ADDX	Sumar con acarreo
LDUH	Cargar media palabra sin signo	ADDXCC	Sumar con acarreo, ajustar icc
LD	Cargar palabra	SUB	Restar
LDD	Cargar doble palabra	SUBCC	Restar, ajustar icc
STB	Almacenar byte	SUBX	Restar con adeudo
STH	Almacenar media palabra	SUBXCC	Restar con adeudo, ajustar icc
STD	Almacenar palabra	MULSCC	Paso de multiplicación, ajustar icc
STDD	Almacenar doble palabra	**Instrucciones de salto/bifurcación**	
Instrucciones de desplazamiento		BCC	Bifurcar si condición
SLL	Desplazamiento lógico a la izquierda	FBCC	Bifurcar si condición de coma flotante
SRL	Desplazamiento lógico a la derecha	CBCC	Bifurcar si condición del coprocesador
SRA	Desplazamiento aritmético a la derecha	CALL	Llamar a procedimiento
Instrucciones booleanas		JMPL	Saltar y enlazar
AND	Y	TCC	Interceptar si condición
ANDCC	Y, ajustar icc	SAVE	Avanzar ventana de registros
ANDN	NO Y	RESTORE	Mover ventanas hacia atrás
ANDNCC	NO Y, ajustar icc	RETT	Retornar de interceptación
OR	O	**Instrucciones diversas**	
ORCC	O, ajustar icc	SETHI	Fijar los 22 bits altos
ORN	NO O	UNIMP	Instrucción no implementada (interceptación)
ORNCC	NO O, ajustar icc	RD	Leer un registro especial
XOR	O exclusiva	WR	Escribir en un registro especial
XORCC	O exclusiva, ajustar icc	IFLUSH	Vaciar la caché de instrucciones
XNOR	NO O exclusiva		
XNORCC	NO O exclusiva, ajustar icc		

El único modo de direccionamiento disponible, aparte del modo registro, es el modo de desplazamiento. Esto es, la dirección efectiva de un operando consiste en una dirección contenida en un registro más un desplazamiento:

$$EA = (R_{S1}) + S2$$
$$\text{o} \quad EA = (R_{S1}) + (R_{S2})$$

dependiendo de si el segundo operando es un dato inmediato o una referencia a registro. Para realizar una carga o un almacenamiento se añade una etapa extra al ciclo de instrucción. Durante la segunda etapa, la dirección de memoria se calcula usando la ALU; la carga o almacenamiento tiene lugar en la tercera etapa. Este modo de direccionamiento sencillo es bastante versátil y puede usarse para sintetizar otros modos de direccionamiento, como se indica en la Tabla 13.13.

Tabla 13.13. Síntesis de otros modos de direccionamiento a partir de los del SPARC.

Modo	Algoritmo	Equivalente del SPARC	Tipo de instrucción
Inmediato	operando = A	S2	Registro a registro
Directo	EA = A	R_0 + S2	Carga, almacenamiento
Registro	EA = R	R_{S1}, R_{S2}	Registro a registro
Indirecto a través de registro	EA = (R)	R_{S1} + 0	Carga, almacenamiento
Desplazamiento	EA = (R) + A	R_{S1} + S2	Carga, almacenamiento

Es instructivo comparar la capacidad de direccionamiento del SPARC con la del MIPS. El MIPS utiliza un desplazamiento de 16 bits, mientras que el SPARC utiliza 13 bits. Por otra parte, el MIPS no permite que se construya una dirección a partir de los contenidos de dos registros.

FORMATO DE INSTRUCCIÓN

Como el MIPS R4000, el SPARC emplea un conjunto sencillo de formatos de instrucción de 32 bits (Figura 13.13). Todas las instrucciones comienzan con un código de operación de dos bits. En ciertas instrucciones, este código se amplía con bits de código de operación adicionales en otra parte del formato. En la instrucción de llamada, un operando inmediato de treinta bits se amplía con dos bits a cero a la derecha, para formar una dirección de 32 bits relativa a PC en complemento a dos. Las instrucciones se alinean en límites de 32 bits por lo que basta esta forma de direccionamiento.

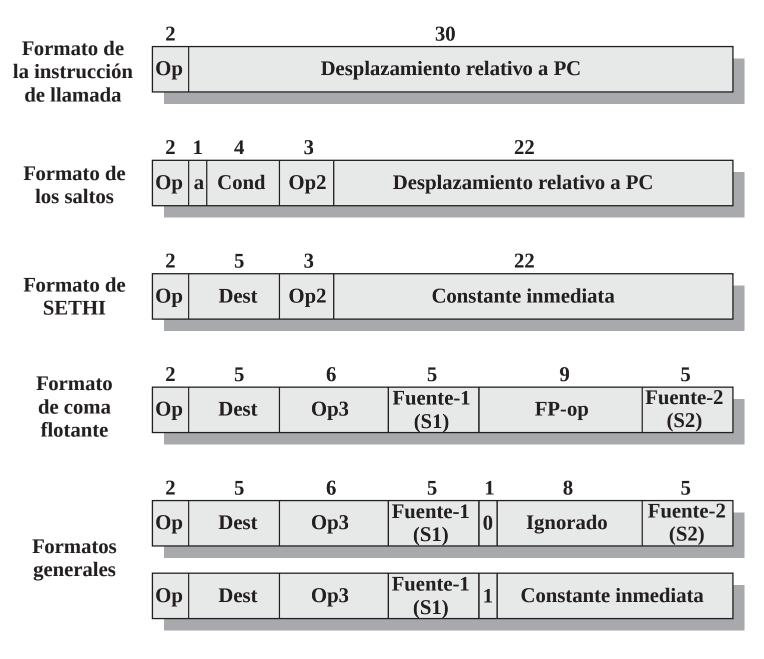


Figura 13.13. Formatos de instrucción del SPARC.

La instrucción de bifurcación incluye un campo de condición de cuatro bits que corresponden a los bits de códigos de condición normales, de modo que puede comprobarse cualquier combinación de condiciones. A la dirección de 22 bits relativa a PC se añaden dos bits a cero por la derecha para formar una dirección relativa de 24 bits en complemento a dos. Una característica poco frecuente de la instrucción de salto es el bit de anulación. Cuando el bit de anulación no está a uno, la instrucción después de la de salto siempre se ejecuta, sin tener en cuenta si se produce el salto. Esta es la típica operación de salto retardado que se encuentra en muchas máquinas RISC y que se describe en la Sección 13.5 (*ver* Figura 13.7). Sin embargo, cuando el bit de anulación está a uno, solo se ejecuta la instrucción siguiente a la de bifurcación si se produce el salto. El procesador puede suprimir el efecto de esta instrucción aunque ya esté en el cauce. Este bit de anulación es útil porque hace más fácil al compilador la tarea de rellenar el espacio de retardo que sigue a un salto condicional. La instrucción destino del salto siempre se puede poner en el ciclo de retardo, ya que si no se produce el salto, esa instrucción puede anularse. El motivo de que esta técnica sea conveniente es que en las bifurcaciones condicionales generalmente se produce el salto más de la mitad de las veces.

La instrucción SETHI es una instrucción especial que se usa para cargar o almacenar un valor de 32 bits. Es necesaria para cargar y almacenar constantes grandes y direcciones. La instrucción SETHI asigna los 22 bits de su operando inmediato a los 22 bits de orden superior de un registro, y rellena con ceros los diez bits de orden inferior. En uno de los formatos generales puede especificarse una constante de hasta trece bits, y esa instrucción podría usarse para rellenar los restantes diez bits del registro. También puede usarse una instrucción de carga o almacenamiento para conseguir un modo de direccionamiento directo. Para cargar un valor de la posición K de memoria, podríamos usar las siguientes instrucciones del SPARC:

```
sethi   %hi(K), %r8            cargar los 22 bits más altos de la dirección
        ;de K en el registro r8
ld      [%r8 + %lo(K)], %r8    cargar el contenido de la posición K en r8
```

Las macros %hi y %lo se usan para definir operandos inmediatos que consistan en los bits de dirección adecuados de una posición de memoria. Este uso de SETHI es similar al de la instrucción LUI del MIPS (Tabla 13.10).

El formato de coma flotante se usa para operaciones en coma flotante. Se designan dos registros fuente y uno destino.

Por último, todas las demás operaciones, que incluyen cargas, almacenamientos, operaciones aritméticas y operaciones lógicas, usan uno de los dos últimos formatos que se muestran en la Figura 13.13. Uno de los formatos utiliza dos registros fuente y uno destino, mientras que el otro utiliza un registro fuente, un operando inmediato de 13 bits, y un registro destino.

13.8. LA CONTROVERSIA ENTRE RISC Y CISC

Durante muchos años, la tendencia general en la arquitectura y organización de computadores ha sido incrementar la complejidad del procesador: más instrucciones, más modos de direccionamiento, más registros especializados, etc. El movimiento RISC representa una ruptura fundamental con la filosofía que hay detrás de esa tendencia. Naturalmente, la aparición de los sistemas RISC y la publicación

de artículos por parte de sus defensores ensalzando las virtudes de los RISC, condujo a una reacción de los implicados en el diseño de arquitecturas CISC.

El trabajo que se ha hecho para evaluar las ventajas de la aproximación RISC puede agruparse en dos categorías:

- **Cuantitativa:** intentos de comparar el tamaño de los programas y su velocidad de ejecución en máquinas RISC y CISC de similar tecnología.
- **Cualitativa:** revisión de asuntos tales como el soporte de lenguajes de alto nivel y el uso óptimo de los recursos VLSI.

La mayoría del trabajo de la evaluación cuantitativa lo han hecho aquellos que trabajan en sistemas RISC [PATT82b, HEAT84, PATT84] y ha sido, por lo general, favorable a la aproximación RISC. Hay otros que han examinado este asunto y no han terminado de convencerse [COLW85a, FLYN87, DAVI87]. Cuando se intentan realizar tales comparaciones surgen varios problemas [SERL86]:

- No hay una pareja de máquinas RISC y CISC que sean comparables en cuanto a coste del ciclo de vida, nivel de tecnología, complejidad a nivel de puertas, sofisticación del compilador, soporte para el sistema operativo, etc.
- No existe un conjunto de programas de prueba definitivo. Las prestaciones varían según el programa.
- Es difícil separar los efectos del hardware de los efectos debidos a la habilidad en el diseño del compilador.
- La mayor parte de los análisis comparativos con RISC se han hecho con máquinas «de juguete» en vez de con productos comerciales. Además, la mayoría de las máquinas comerciales anunciadas como RISC poseen una mezcla de características RISC y CISC. Por tanto, es difícil una comparación equitativa con una máquina CISC comercial «pura» (por ejemplo, VAX, Pentium).

La valoración cualitativa es, casi por definición, subjetiva. Varios investigadores han fijado su atención en tal valoración [COLW85a, WALL85] pero los resultados son, en el mejor de los casos, ambiguos, claramente susceptibles de refutación [PATT85b] y, por supuesto, de contrarrefutación [COLW85b].

En los años más recientes, la controversia RISC frente a CISC se ha sosegado en gran parte. Ello se debe a que ha habido una convergencia progresiva de las tecnologías. Conforme la densidad de integración y la velocidad bruta del hardware han aumentado, los sistemas RISC se han vuelto más complejos. Al mismo tiempo, en un esfuerzo por exprimir las prestaciones al máximo, los diseños CISC se han concentrado en cuestiones asociadas tradicionalmente a los RISC, tales como un mayor número de registros de propósito general y un énfasis creciente en el diseño del cauce de instrucciones.

13.9. LECTURAS RECOMENDADAS

Dos artículos clásicos que ofrecen una visión de conjunto sobre RISC son [PATT85a] y [HENN84]. Otro artículo que hace una descripción general es [STAL88]. [RADI83] y [PATT82a] ofrecen informes sobre dos esfuerzos pioneros en RISC.

[KANE92] trata en detalle la máquina comercial MIPS. [MIRA92] proporciona una buena visión general del MIPS R4000. [BASH91] discute la evolución desde la segmentación del R3000 hasta la supersegmentación del R4000. [DEWA90] cubre el SPARC con cierto detalle.

BASH91 Bashteen, A.; Lui, I. y Mullan, J.: «A Superpipeline Approach to the MIPS Architecture». *Proceedings, COMPCON Spring '91,* febrero, 1991.

DEWA90 Dewar, R. y Smosna, M.: Microprocessors: *A Programmer's View.* New York. McGraw-Hill, 1990.

HENN84 Hennesy, J.: «VLSI Processor Architecture». *IEEE Transactions on Computers,* diciembre, 1984.

KANE92 Kane, G. y Heinrich, J.: MIPS RISC *Architecture.* Englewood Cliffs, NJ. Prentice-Hall, 1992.

MIRA92 Mirapuri, S.; Woodacre, M. y Vasseghi, N.: «The MIPS R4000 Processor». *IEEE Micro,* abril, 1992.

PATT82a Patterson, D. y Sequin, C.: «A VLSI RISC». *Computer,* septiembre, 1982.

PATT85a Patterson, D.: «Reduced Instruction Set Computers». *Communications of the ACM,* enero, 1985.

RADI83 Radin, G.: «The 801 Minicomputer». *IBM Journal of Research and Development,* mayo, 1983.

STAL88 Stallings, W.: «Reduced Instruction Set Computer Architecture». *Proceedings of the IEEE,* enero, 1988.

13.10 PALABRAS CLAVE, PREGUNTAS DE REPASO Y PROBLEMAS

PALABRAS CLAVE

banco de registros carga retardada	computador de repertorio complejo de instrucciones (CISC) computador de repertorio reducido de instrucciones (RISC)	lenguaje de alto nivel (HLL) salto retardado SPARC ventana de registros

PREGUNTAS DE REPASO

13.1. ¿Cuáles son las algunas de las características típicas que distinguen la organización RISC?

13.2. Explique brevemente las dos soluciones básicas empleadas para minimizar las operaciones registro-memoria en las máquinas RISC.

13.3. Si se usa un buffer circular de registros para manejar las variables locales en procedimientos anidados, describa dos aproximaciones para gestionar las variables globales.

13.4. ¿Cuáles son algunas de las características típicas de una arquitectura de repertorio de instrucciones de tipo RISC?

13.5. ¿Qué es un salto retardado?

PROBLEMAS

13.1. Considere el patrón de llamada-retorno de la Figura 4.16, ¿cuántos desbordamientos y desbordamientos hacia cero (cada uno de los cuales causa una salvaguarda/restauración de registros) ocurrirán con un tamaño de ventana de

(a) 5?
(b) 8?
(c) 16?

13.2. En la discusión de la Figura 13.2 se explicó que solo las dos primeras partes de una ventana se guardan o se restauran. ¿Por qué no es necesario guardar los registros temporales?

13.3. Queremos determinar el tiempo de ejecución de un programa dado usando los diversos esquemas de segmentación estudiados en la Sección 13.5. Sean

N = número de instrucciones ejecutadas
D = número de accesos a memoria
J = número de instrucciones de salto

Para el sencillo esquema secuencial (Figura 13.6a), el tiempo de ejecución es $2N + D$ etapas. Obtenga las fórmulas del tiempo de ejecución para la segmentación en dos etapas, tres etapas y cuatro etapas.

13.4. Reorganice la secuencia de código de la Figura 13.6d para reducir el número de instrucciones NOOP.

13.5. Considere el siguiente fragmento de código en un lenguaje de alto nivel:

```
for I in 1...100 loop
  S ← S + Q(I).VAL
end loop;
```

Suponga que Q es una matriz de registros de 32 bytes y que el campo VAL está en los primeros cuatro bytes de cada registro. Usando código del 80x86, podemos compilar este fragmento de programa como sigue:

```
        MOV     ECX,1           ;usar el registro ECX para contener I
LP:     IMUL    EAX, ECX, 32    ;poner el desplazamiento en EAX
        MOV     EBX, Q[EAX]     ;cargar el campo VAL
        ADD     S, EBX          ;sumar a S
        INC     ECX             ;incrementar I
        CMP     ECX, 101        ;comparar con 101
        JNE     LP              ;seguir en el bucle hasta que I = 100
```

Este programa utiliza la instrucción IMUL, que multiplica el segundo operando por el valor inmediato del tercer operando y lleva el resultado al primer operando (ver Problema 10.13). A un defensor de los RISC le gustaría demostrar que un compilador ingenioso puede eliminar instrucciones complejas innecesarias tales como IMUL. Dé una demostración escribiendo de nuevo el programa para 80x86 anterior sin usar la instrucción IMUL.

13.6. Considere el siguiente bucle:

```
S := 0;
for K := 1 to 100 do
      S := S − K;
```

Un traducción directa de este bucle a un lenguaje ensamblador genérico se parecería a algo como esto:

```
        LD      R1, 0           ;almacenar el valor de S en R1
        LD      R2, 1           ;almacenar el valor de K en R2
LP      SUB     R1, R1, R2      ;S := S − K
        BEQ     R2, 100, EXIT   ;salir del bucle si K = 100
        ADD     R2, R2, 1       ;si no, incrementar K
        JMP     LP              ;volver al principio del bucle
```

Un compilador de una máquina RISC introducirá ciclos de retardo en este código para que el procesador pueda emplear el mecanismo de salto retardado. La instrucción JMP es fácil de tratar, porque siempre va seguida de la instrucción SUB; por tanto, podemos simplemente poner una copia de la instrucción SUB en el espacio de retardo tras la instrucción JMP. La instrucción BEQ plantea una dificultad. No podemos dejar el código como está, porque la instrucción ADD se ejecutaría una vez más de la cuenta. Por consiguiente, hace falta una instrucción NOP. Muestre el código resultante.

13.7. Una máquina RISC puede realizar tanto una asignación de registros simbólicos a registros reales como una reorganización de las instrucciones para aumentar la eficiencia de la segmentación. Surge una interesante cuestión sobre el orden en el que deben realizarse esas dos operaciones. Considere el siguiente fragmento de programa:

```
LD    SR1, A            cargar A en el registro simbólico 1
LD    SR2, B;           cargar B en el registro simbólico 2
ADD   SR3, SR1, SR2     sumar los contenidos de SR1 y SR2 en SR3
LD    SR4, C
LD    SR5, D
ADD   SR6, SR4, SR5
```

(a) Realice en primer lugar la asignación de registros y después algunas de las posibles reordenaciones de instrucciones. ¿Cuántos registros de la máquina se usan? ¿Ha habido alguna mejora en la segmentación?

(b) .Partiendo del programa original, realice ahora la reordenación de instrucciones y después alguna de las posibles asignaciones. ¿Cuántos registros de la máquina se usan? ¿Ha habido alguna mejora en la segmentación?

13.8. Añada filas para los siguientes procesadores en la Tabla 13.7:

(a) Pentium II
(b) PowerPC

13.9. En muchos casos, las instrucciones máquina usuales que no se incluyen como parte del repertorio de instrucciones del MIPS pueden sintetizarse con una única instrucción del MIPS. Muéstrelo para las siguientes:

(a) Transferencia registro a registro
(b) Incrementar, decrementar
(c) Complementar
(d) Negar
(e) Poner a cero

13.10. Una implementación del SPARC tiene *K* ventanas de registros. ¿Cuál es el número de registros físicos *N*?

13.11. El SPARC carece de varias instrucciones que se encuentran generalmente en las máquinas CISC. Algunas de ellas se simulan fácilmente usando el registro R0, que siempre vale 0, o un operando constante. Estas instrucciones simuladas se llaman pseudoinstrucciones y el compilador del SPARC las reconoce. Muestre cómo simular las pseudoinstrucciones siguientes con una única instrucción del SPARC. En todas ellas, «fuen» y «dest» se refieren a registros. *Pista:* un almacenamiento en R0 no tiene efecto.

(a) MOV fuen, dest	(d) NOT dest	(g) DEC dest
(b) COMPARE fuen1, fuen2	(e) NEG dest	(h) CLR dest
(c) TEST fuen1	(f) INC dest	(i) NOP

13.12. Considere el fragmento de código siguiente:

```
if K > 10
   L: = K + 1
else
   L: = K − 1;
```

Una traducción directa de esta sentencia a ensamblador de SPARC podría tener la siguiente forma:

```
sethi   %hi(K), %r8            cargar los 22 bits altos de la dirección
                               de K en el registro r8
ld      [%r8 + %lo(K)], %r8    cargar contenido de la posición K en r8
cmp     %r8, 10                comparar contenido de r8 con 10
```

```
        ble     L1                        saltar si (r8) ≤ 10
        nop
        sethi   %hi(K), %r9
        ld      [%r9 + %lo(K)], %r9       cargar contenido de la posición K en r9
        inc     %r9                       sumar 1 a (r9)
        sethi   %hi(L), %r10
        st      %r9, [%r10 + %lo(L)]      almacenar (r9) en la posición L
        bL2
        nop
L1:     sethi   %hi(K), %r11
        ld      [%r11 + %lo(K)], %r12     cargar contenido de la posición K en r12
        dec     %r12                      restar uno a (r12)
        sethi   %hi(L), %r13
        st      %r12, [%r13 + %lo(L)]     almacenar (r12) en la posición L
L2:
```

El código contiene un «nop» después de cada instrucción de salto para permitir la operación de salto retardado.

(a) Las optimizaciones de un compilador normal que no tienen nada que hacer en las máquinas RISC son, en general, efectivas para poder realizar dos transformaciones en el código precedente. Observe que dos de las cargas no son necesarias y que los dos almacenamientos se pueden unir si el almacenamiento se mueve a otra posición dentro del código. Muestre el programa después de hacer estos dos cambios.

(b) Es posible ahora realizar algunas optimizaciones propias del SPARC. El «nop» después del «ble» puede reemplazarse moviendo otra instrucción a ese espacio de retardo y activando el bit de anulación en la instrucción «ble» (expresado como ble,a L1). Muestre el programa tras este cambio.

(c) Ahora hay dos instrucciones innecesarias. Elimínelas y muestre el programa resultante.

CAPÍTULO 14

Paralelismo en las instrucciones y procesadores superescalares

PUNTOS CLAVE

- Un **procesador superescalar** es aquél que usa múltiples cauces de instrucciones independientes. Cada cauce consta de múltiples etapas, de modo que puede tratar varias instrucciones a la vez. El hecho de que haya varios cauces introduce un nuevo nivel de paralelismo, permitiendo que varios flujos de instrucciones se procesen simultáneamente. Un procesador superescalar saca provecho de lo que se conoce como **paralelismo en las instrucciones**, que hace referencia al grado en que las instrucciones de un programa pueden ejecutarse en paralelo.
- Típicamente, un procesador superescalar capta varias instrucciones a la vez, y a continuación intenta encontrar instrucciones cercanas que sean independientes entre sí y puedan, por consiguiente, ejecutarse en paralelo. Si la entrada de una instrucción depende de la salida de una instrucción precedente, la segunda instrucción no puede completar su ejecución al mismo tiempo o antes que la primera. Una vez que se han identificado tales dependencias, el procesador puede emitir y completar instrucciones en un orden diferente al del código máquina original.
- El procesador puede eliminar algunas dependencias innecesarias mediante el uso de registros adicionales y el renombramiento de las referencias a registros del código original.
- Mientras que los procesadores RISC puros con frecuencia emplean saltos retardados para maximizar la utilización del cauce de instrucciones, este método es menos apropiado para las máquinas superescalares. En lugar de eso, la mayoría de las máquinas superescalares emplean métodos tradicionales de predicción de saltos para aumentar su rendimiento.

Una implementación superescalar de la arquitectura de un procesador es aquella en la que las instrucciones comunes —aritmética entera y de coma flotante, cargas, almacenamientos y saltos condicionales— pueden iniciar su ejecución simultáneamente y ejecutarse de manera independiente. Estas implementaciones plantean complejos problemas de diseño relacionados con el cauce de instrucciones.

El diseño superescalar aparece en escena muy cerca de la arquitectura RISC. Aunque la arquitectura de repertorio de instrucciones simplificado de una máquina RISC se preste fácilmente a utilizar técnicas superescalares, la aproximación superescalar se puede usar tanto en una arquitectura RISC como en una CISC.

Mientras el periodo de gestación desde el comienzo de la auténtica investigación en RISC, con el IBM 801 y el RISC I de Berkeley, hasta la llegada de máquinas RISC comerciales fue de siete u ocho años, las primeras máquinas superescalares estuvieron disponibles comercialmente tan solo un año o dos después de que se acuñara el término *superescalar*. La aproximación superescalar ha llegado a convertirse en el método habitual de implementación de microprocesadores de altas prestaciones.

En este capítulo, comenzamos con una visión de conjunto de la aproximación superescalar, contrastándola con la supersegmentación. Después se presentan las cuestiones de diseño más importantes relacionadas con la implementación superescalar. Más adelante estudiamos varios ejemplos importantes de arquitecturas superescalares.

14.1. VISIÓN DE CONJUNTO

El término *superescalar*, acuñado en 1987 [AGER87], hace referencia a una máquina diseñada para mejorar la velocidad de ejecución de las instrucciones escalares. En la mayoría de las aplicaciones, la mayor parte de las operaciones se realizan con cantidades escalares. Así pues, la aproximación superescalar representa el siguiente paso en la evolución de los procesadores de uso general y altas prestaciones.

Lo esencial del enfoque superescalar es su habilidad para ejecutar instrucciones en diferentes cauces de manera independiente y concurrente. El concepto puede llevarse más lejos permitiendo que las instrucciones se ejecuten en un orden diferente al del programa. La Figura 14.1 muestra, en términos generales, el planteamiento superescalar. Hay múltiples unidades funcionales, cada una de las cuales está implementada como un cauce segmentado, que admiten la ejecución en paralelo de varias instrucciones. En el ejemplo, dos operaciones enteras, dos de coma flotante y una de memoria (carga o almacenamiento) pueden estar ejecutándose en el mismo instante.

Muchos investigadores han estudiado procesadores de tipo superescalar, y su investigación indica que es posible cierto grado de mejora de las prestaciones. La Tabla 14.1 presenta las mejoras en

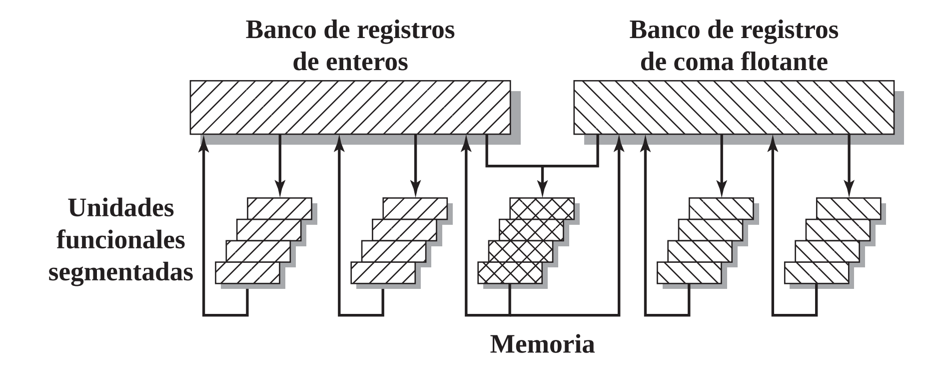


Figura 14.1. Organización superescalar general [COME95].

Tabla 14.1. Modos de direccionamiento del Pentium.

Referencia	Incremento de velocidad
[TJAD70]	1,8
[KUCK72]	8
[WEIS84]	1,58
[ACOS86]	2,7
[SOHI90]	1,8
[SMIT89]	2,3
[JOUP89b]	2,2
[LEE91]	7

velocidad reseñadas. Las diferencias en los resultados se deben tanto a las diferencias en el hardware de las máquinas simuladas, como a las de las aplicaciones ejecutadas.

SUPERESCALAR FRENTE A SUPERSEGMENTADO

Una solución alternativa para alcanzar mayores prestaciones es la llamada supersegmentación, un término acuñado en 1988 [JOUP88]. La supersegmentación aprovecha el hecho de que muchas etapas del cauce realizan tareas que requieren menos de medio ciclo de reloj. De este modo, doblando la velocidad de reloj interna se permite la realización de dos tareas en un ciclo de reloj externo. Hemos visto un ejemplo de esta aproximación en el MIPS R4000.

La Figura 14.2 compara las dos aproximaciones. La parte superior del diagrama ilustra un cauce normal, usado como base de la comparación. El cauce base emite una instrucción por ciclo de reloj y puede ejecutar una etapa del cauce en cada ciclo. El cauce tiene cuatro etapas: captación de instrucción, decodificación de la operación, ejecución de la operación y escritura del resultado. La etapa de ejecución se ha destacado con una trama por motivos de claridad. Observe que aunque se ejecuten varias instrucciones concurrentemente, solo hay una instrucción en la etapa de ejecución en un determinado instante.

La parte siguiente del diagrama muestra una implementación supersegmentada que es capaz de ejecutar dos etapas del cauce por ciclo de reloj. Una forma alternativa de enfocar esto consiste en que las funciones realizadas en cada etapa se pueden dividir en dos partes no solapadas y que cada una se ejecuta en medio ciclo de reloj. Se dice que una implementación de un cauce supersegmentado que se comporta de esta forma es de grado 2. Por último, la parte inferior del diagrama muestra una implementación superescalar capaz de ejecutar en paralelo dos instrucciones en cada etapa. Naturalmente, también son posibles implementaciones supersegmentadas y superescalares de mayor grado.

Las dos realizaciones, supersegmentada y superescalar, representadas en la Figura 14.2 ejecutan el mismo número de instrucciones en el mismo tiempo cuando funcionan de forma ininterrumpida. El procesador supersegmentado se queda atrás con respecto al procesador superescalar al comienzo del programa y en cada destino de un salto.

LIMITACIONES

La aproximación superescalar depende de la habilidad para ejecutar múltiples instrucciones en paralelo. La expresión **paralelismo en las instrucciones** se refiere al grado en el que, en promedio, las instrucciones de un programa se pueden ejecutar en paralelo. Para maximizar el paralelismo en las instrucciones, se puede usar una combinación de optimizaciones realizadas por el compilador y de técnicas hardware. Antes de examinar las técnicas de diseño utilizadas en las máquinas superescalares para aumentar el paralelismo en las instrucciones, debemos considerar las limitaciones fundamentales del paralelismo a las que el sistema tiene que enfrentarse. [JOHN91] enumera cinco limitaciones:

- Dependencia de datos verdadera.
- Dependencia relativa al procedimiento.
- Conflicto en los recursos.
- Dependencia de salida.
- Antidependencia.

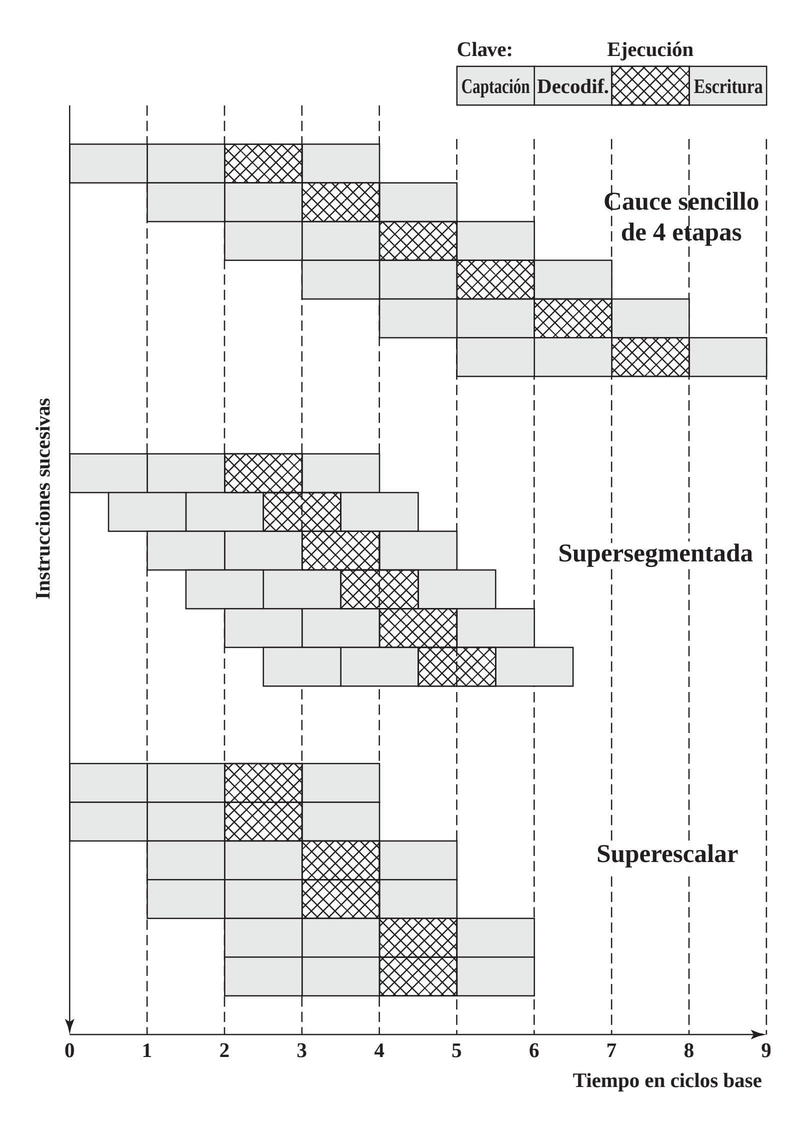


Figura 14.2. Comparación de las aproximaciones superescalar y supersegmentada.

En lo que resta de esta sección examinamos las tres primeras limitaciones. El estudio de las dos últimas debe esperar a algunos desarrollos de la siguiente sección.

Dependencia de datos verdadera. Consideremos la siguiente secuencia[1]:

```
add     r1, r2   ;cargar el registro r1 con el contenido de r2 más
                  el contenido de r1
move    r3, r1   ;cargar el registro r3 con el contenido de r1
```

[1] En el lenguaje ensamblador de Intel 80x86 y Pentium, un comentario se indica mediante un punto y coma. El ensamblador ignora el punto y coma y todos los caracteres posteriores de la misma línea.

La segunda instrucción se puede captar y decodificar, pero no se puede ejecutar hasta que finalice la ejecución de la primera instrucción. El motivo es que la segunda instrucción necesita un dato producido por la primera instrucción. Esta situación es conocida como **dependencia de datos verdadera** (también llamada **dependencia de flujo** o **dependencia escritura-lectura**).

La Figura 14.3 ilustra esta dependencia en una máquina superescalar de grado 2. Si no hay dependencias, se puede captar y ejecutar dos instrucciones en paralelo. En caso de que exista dependencia de datos entre la primera y la segunda instrucción, se retrasa la segunda instrucción tantos ciclos de reloj como sea necesario para eliminar la dependencia. En general, cualquier instrucción debe retrasarse hasta que todos sus valores de entrada estén disponibles.

En un cauce escalar simple, como el ilustrado en la parte superior de la Figura 14.2, la secuencia de instrucciones anterior no causaría ningún retraso. Sin embargo, consideremos la siguiente secuencia, en la cual una de las cargas se hace desde la memoria y no desde un registro:

```
load    r1, ef    ;cargar el registro r1 con el contenido de la
                   dirección de memoria efectiva ef
move    r3, r1    ;cargar el registro r3 con el contenido de r1
```

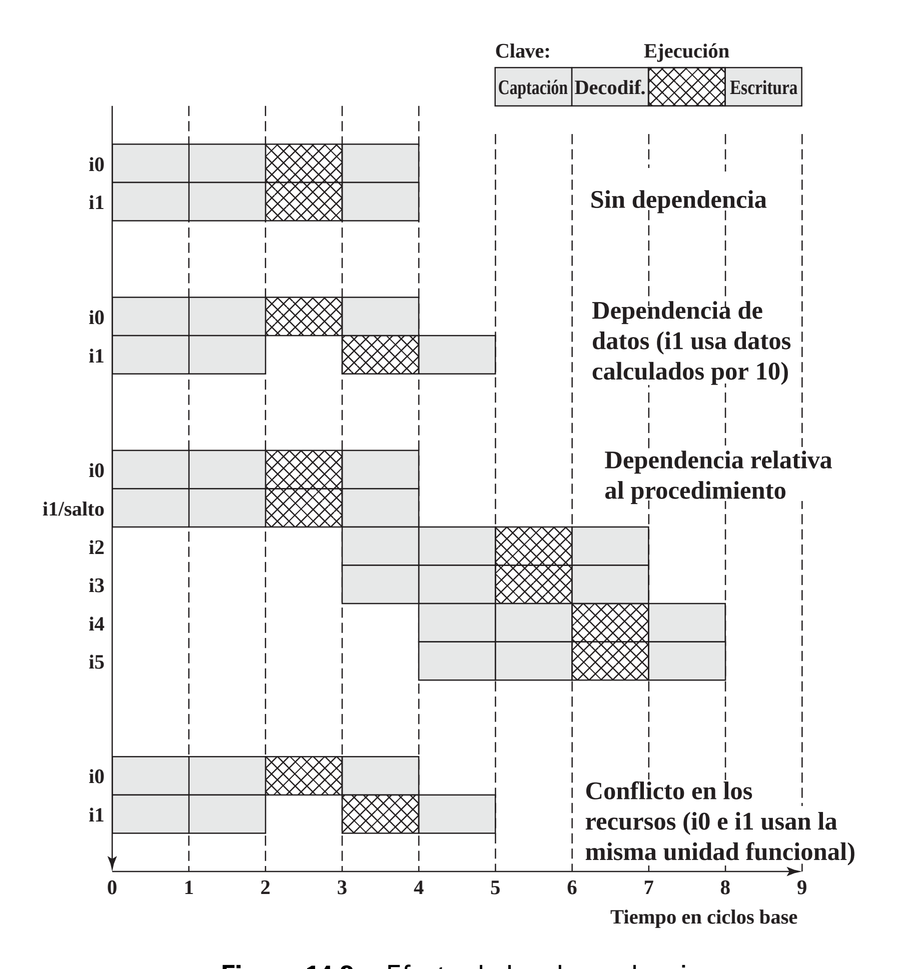


Figura 14.3. Efecto de las dependencias.

Un procesador RISC típico tarda dos o más ciclos en realizar una carga desde memoria debido al tiempo de acceso a memoria o caché externas al chip. Una forma de compensar este retraso consiste en que el compilador reordene las instrucciones de tal modo que una o más instrucciones posteriores que no dependan de la carga desde memoria puedan empezar a fluir a través del cauce. Este esquema es menos efectivo en el caso de un cauce superescalar: las instrucciones independientes que se ejecutan durante la carga lo hacen probablemente en el primer ciclo de esta, dejando al procesador sin nada que hacer hasta que concluya la carga.

Dependencias relativas al procedimiento. Según se discutió en el Capítulo 12, la presencia de saltos en una secuencia de instrucciones complica el funcionamiento del cauce. Las instrucciones que siguen a una bifurcación (en la que se puede saltar o no) tienen una dependencia relativa al procedimiento en esa bifurcación y no pueden ejecutarse hasta que se ejecute el salto. La Figura 14.3 ilustra el efecto de un salto en un cauce superescalar de grado 2.

Como hemos visto, este tipo de dependencia relativa al procedimiento también afecta a un cauce escalar. Las consecuencias para un cauce superescalar son más graves, ya que se pierde un mayor número de oportunidades de comenzar a ejecutar instrucciones en cada retardo.

Si se usan instrucciones de longitud variable, surge otro tipo de dependencia relativa al procedimiento. Puesto que no se conoce la longitud de una instrucción concreta, esta ha de decodificarse al menos parcialmente antes de captar la siguiente instrucción. Ello impide la captación simultánea necesaria en un cauce superescalar. Esta es una de las razones por las que las técnicas superescalares se aplican más fácilmente a arquitecturas RISC o similares, que tienen una longitud de instrucción fija.

Conflicto en los recursos. Un conflicto en un recurso es una pugna de dos o más instrucciones por el mismo recurso al mismo tiempo. Ejemplos de recursos son las memorias, las cachés, los buses, los puertos del banco de registros y las unidades funcionales (por ejemplo, un sumador de la ALU).

Desde el punto de vista del cauce segmentado, un conflicto en los recursos presenta el mismo comportamiento que una dependencia de datos (Figura 14.3). No obstante, hay algunas diferencias. Por un lado, los conflictos en los recursos pueden superarse duplicando estos, mientras que una dependencia de datos verdadera no se puede eliminar. Además, cuando una operación tarda mucho tiempo en finalizar, los conflictos en los recursos se pueden minimizar segmentando la unidad funcional apropiada.

14.2. CUESTIONES RELACIONADAS CON EL DISEÑO

PARALELISMO EN LAS INSTRUCCIONES Y PARALELISMO DE LA MÁQUINA

[JOUP89a] hace una importante distinción entre dos conceptos relacionados: el paralelismo en las instrucciones y el paralelismo de la máquina. Existe **paralelismo en las instrucciones** cuando las instrucciones de una secuencia son independientes y por tanto pueden ejecutarse en paralelo solapándose.

Como ejemplo del concepto de paralelismo en las instrucciones, consideremos los dos siguientes fragmentos de código [JOUP89b]:

```
Load R1 ← R2            Add R3 ← R3, "1"
Add R3 ← R3, "1"        Add R4 ← R3, R2
Add R4 ← R4, R2         Store [R4] ← R0
```

Las tres instrucciones de la izquierda son independientes, y en teoría las tres podrían ejecutarse en paralelo. Por contra, las tres instrucciones de la derecha no pueden ejecutarse en paralelo porque la segunda instrucción usa el resultado de la primera, y la tercera instrucción usa el resultado de la segunda.

El paralelismo en las instrucciones depende de la frecuencia de dependencias de datos verdaderas y dependencias relativas al procedimiento que haya en el código. Estos factores dependen a su vez de la arquitectura del repertorio de instrucciones y de la aplicación. El paralelismo en las instrucciones depende también de lo que [JOUP89a] llama latencia de una operación: el tiempo que transcurre hasta que el resultado de una instrucción está disponible para ser usado como operando de una instrucción posterior. La latencia determina cuánto retardo causará una dependencia de datos o relativa al procedimiento.

El **paralelismo de la máquina** es una medida de la capacidad del procesador para sacar partido al paralelismo en las instrucciones. El paralelismo de la máquina depende del número de instrucciones que pueden captarse y ejecutarse al mismo tiempo (número de cauces paralelos) y de la velocidad y sofisticación de los mecanismos que usa el procesador para localizar instrucciones independientes.

Tanto el paralelismo en las instrucciones como el paralelismo de la máquina son factores importantes para mejorar las prestaciones. Un programa puede no tener el suficiente nivel de paralelismo en las instrucciones como para sacar el máximo partido al paralelismo de la máquina. El empleo de una arquitectura con instrucciones de longitud fija, como en un RISC, aumenta el paralelismo en las instrucciones. Por otra parte, un escaso paralelismo de la máquina limitará las prestaciones sin que importe la naturaleza del programa.

POLÍTICAS DE EMISIÓN DE INSTRUCCIONES

Como mencionamos antes, el paralelismo de la máquina no solo es un asunto relacionado con la existencia de múltiples réplicas de cada etapa del cauce. El procesador además tiene que ser capaz de identificar el paralelismo en las instrucciones y organizar la captación, decodificación, y ejecución de las instrucciones en paralelo. [JOHN91] utiliza el término **emisión de instrucciones** para referirse al proceso de iniciar la ejecución de instrucciones en las unidades funcionales del procesador y el término **política de emisión de instrucciones** para referirse al protocolo usado para emitir instrucciones. En general, se puede decir que la emisión de una instrucción tiene lugar cuando esta pasa de la etapa de decodificación del cauce a la primera etapa de ejecución.

Esencialmente, el procesador intenta localizar instrucciones más allá del punto de ejecución en curso que puedan introducirse en el cauce y ejecutarse. Con respecto a esto, hay tres ordenaciones importantes:

- El orden en que se captan las instrucciones.

- El orden en que se ejecutan las instrucciones.
- El orden en que las instrucciones actualizan los contenidos de los registros y de las posiciones de memoria.

Cuanto más sofisticado sea el procesador, menos limitado estará por la estrecha relación entre estas ordenaciones. Para optimizar la utilización de los diversos elementos del cauce, el procesador tendrá que alterar uno o más de estos órdenes con respecto al orden que se encontraría en una ejecución secuencial estricta. La única restricción que tiene el procesador es que el resultado debe ser correcto. De este modo, el procesador tiene que acomodar las diversas dependencias y conflictos discutidos antes.

En términos generales, podemos agrupar las políticas de emisión de instrucciones de los procesadores superescalares en las siguientes categorías:

- Emisión en orden y finalización en orden.
- Emisión en orden y finalización desordenada.
- Emisión desordenada y finalización desordenada.

Emisión en orden y finalización en orden. La política de emisión de instrucciones más sencilla es emitir instrucciones en el orden exacto en que lo haría una ejecución secuencial (emisión en orden) y escribir los resultados en ese mismo orden (finalización en orden). Ni siquiera los cauces escalares siguen una política tan ingenua. No obstante, es útil considerar esta política como base con la cual comparar otras aproximaciones más sofisticadas.

La Figura 14.4a ofrece un ejemplo de esta política. Suponemos un cauce superescalar capaz de captar y decodificar dos instrucciones a la vez, con tres unidades funcionales independientes (por ejemplo, dos de aritmética entera y una de aritmética en coma flotante), y con dos copias de la etapa de escritura del cauce. El ejemplo supone las siguientes restricciones para un fragmento de código de seis instrucciones:

- I1 necesita dos ciclos para ejecutarse.
- I3 e I4 compiten por la misma unidad funcional.
- I5 depende de un valor producido por I4.
- I5 e I6 compiten por una unidad funcional.

Las instrucciones se captan de dos en dos y pasan a la unidad de decodificación. Como las instrucciones se captan por parejas, las dos siguientes instrucciones tienen que esperar hasta que la pareja de etapas de decodificación del cauce se encuentre vacía. Para garantizar la finalización en orden, cuando hay una pugna por una unidad funcional o cuando una unidad funcional necesita más de un ciclo para generar un resultado, la emisión de instrucciones se detiene temporalmente.

En este ejemplo, el tiempo transcurrido desde la decodificación de la primera instrucción hasta la escritura de los últimos resultados es de ocho ciclos.

Emisión en orden y finalización desordenada. La finalización desordenada se usa en los procesadores RISC escalares para mejorar la velocidad de las instrucciones que necesitan ciclos. La

Decodificación		Ejecución			Escritura		Ciclo
I1	I2						1
I3	I4	I1	I2				2
I3	I4	I1					3
	I4			I3	I1	I2	4
I5	I6			I4			5
	I6		I5		I3	I4	6
			I6				7
					I5	I6	8

(a) Emisión en orden y finalización en orden

Decodificación		Ejecución			Escritura		Ciclo
I1	I2						1
I3	I4	I1	I2				2
	I4	I1		I3	I2		3
I5	I6			I4	I1	I3	4
	I6		I5		I4		5
			I6		I5		6
					I6		7

(b) Emisión en orden y finalización desordenada

Decodificación		Ventana	Ejecución			Escritura		Ciclo
I1	I2							1
I3	I4	*I1, I2*	I1	I2				2
I5	I6	*I3, I4*	I1		I3	I2		3
		I4, I5, I6		I6	I4	I1	I3	4
		I5		I5		I4	I6	5
						I5		6

(c) Emisión desordenada y finalización desordenada

Figura 14.4. Políticas de emisión y finalización de instrucciones en un cauce superescalar.

Figura 14.4b ilustra su uso en un procesador superescalar. La instrucción I2 se puede ejecutar hasta su conclusión antes de que acabe I1. Ello permite a I3 terminar antes, con el resultado neto de ahorrar un ciclo.

Con la finalización desordenada, puede haber cualquier número de instrucciones en la etapa de ejecución en un momento dado, hasta alcanzar el máximo grado de paralelismo de la máquina ocupando todas las unidades funcionales. La emisión de instrucciones se para cuando hay una pugna por un recurso, una dependencia de datos o una dependencia relativa al procedimiento.

Aparte de las limitaciones anteriores, surge una nueva dependencia, a la cual nos referimos anteriormente como **dependencia de salida** (también llamada **dependencia escritura-escritura**). El siguiente fragmento de código ilustra esta dependencia (*op* representa cualquier operación):

```
I1:  R3 ← R3 op R5
I2:  R4 ← R3 + 1
I3:  R3 ← R5 + 1
I4:  R7 ← R3 op R4
```

La instrucción I2 no puede ejecutarse antes que la instrucción I1, ya que necesita el resultado almacenado en el registro R3 por I1; este es un ejemplo de dependencia de datos verdadera, como se describió en la Sección 14.1. Por la misma razón, I4 debe esperar a I3, ya que usa un resultado producido por esta. ¿Qué ocurre con la relación entre I1 e I3? Aquí no hay dependencia de datos, tal y como la hemos definido. Sin embargo, si I3 se ejecuta hasta el final antes que I1, se captará un valor incorrecto del contenido de R3 para la ejecución de I4. Por consiguiente, I3 debe terminar después de I1 para producir el valor correcto de salida. Para asegurar esto, la emisión de la tercera instrucción debe detenerse si su resultado puede ser sobrescrito más tarde por una instrucción anterior que tarda más en finalizar.

La finalización desordenada necesita una lógica de emisión de instrucciones más compleja que la finalización en orden. Además, es más difícil ocuparse de las interrupciones y excepciones. Cuando ocurre una interrupción, la ejecución de instrucciones se suspende en el punto actual, para reanudarse posteriormente. El procesador debe asegurar que la reanudación tiene en cuenta que, en el momento de la interrupción, algunas instrucciones posteriores a la instrucción que provocó dicha interrupción pueden haber finalizado ya.

Emisión desordenada y finalización desordenada. Con la emisión en orden, el procesador solo decodificará instrucciones hasta el punto de dependencia o conflicto. No se decodifican más instrucciones hasta que el conflicto se resuelve. Por consiguiente, el procesador no puede buscar más allá del punto de conflicto instrucciones que podrían ser independientes de las que hay en el cauce y que podrían introducirse provechosamente en este.

Para permitir la emisión desordenada, es necesario desacoplar las etapas del cauce de decodificación y ejecución. Esto se hace mediante un buffer llamado **ventana de instrucciones**. Con esta organización, cuando un procesador termina de decodificar una instrucción, la coloca en la ventana de instrucciones. Mientras el buffer no se llene, el procesador puede continuar captando y decodificando nuevas instrucciones. Cuando una unidad funcional de la etapa de ejecución queda disponible, se puede emitir una instrucción desde la ventana de instrucciones a la etapa de ejecución. Cualquier instrucción puede emitirse, siempre que (1) necesite la unidad funcional particular que está disponible y (2) ningún conflicto ni dependencia la bloqueen.

El resultado de esta organización es que el procesador tiene capacidad de anticipación, que le permite identificar instrucciones independientes que pueden introducirse en la etapa de ejecución. Las instrucciones se emiten desde la ventana de instrucciones sin que se tenga muy en cuenta su orden original en el programa. Como antes, la única restricción es que el programa funcione correctamente.

La Figura 14.4c ilustra esta política. En cada uno de los tres primeros ciclos, se captan dos instrucciones y se llevan a la etapa de decodificación. En cada ciclo, sujetas a la restricción del tamaño del buffer, se transfieren dos instrucciones desde la etapa de decodificación a la ventana de instrucciones. En este ejemplo, es posible emitir la instrucción I6 delante de la I5 (recuerde que I5 depende de I4, pero I6 no). De este modo se ahorra un ciclo en las etapas de ejecución y de escritura, y el ahorro de principio a fin, comparado con la Figura 14.4b, es de un ciclo.

La ventana de instrucciones se representa en la Figura 14.4c para ilustrar su función. No obstante, esta ventana no es una etapa adicional del cauce. El hecho de que una instrucción esté en la ventana indica sencillamente que el procesador tiene suficiente información sobre esa instrucción como para decidir si puede emitirse.

La política de emisión desordenada y finalización desordenada está sujeta a las mismas restricciones descritas anteriormente. Una instrucción no puede emitirse si viola una dependencia o tiene un conflicto. La diferencia es que ahora hay más instrucciones dispuestas a ser emitidas, reduciendo la probabilidad de que una etapa del cauce tenga que pararse. Además surge una nueva dependencia, a la que nos referimos antes como **antidependencia** (también llamada **dependencia lectura-escritura**). El fragmento de código considerado antes ilustra esta dependencia:

```
I1:  R3 ← R3 op R5
I2:  R4 ← R3 + 1
I3:  R3 ← R5 + 1
I4:  R7 ← R3 op R4
```

La instrucción I3 no puede finalizar antes de que la instrucción I2 comience a ejecutarse y haya captado sus operandos. Esto es así debido a que I3 actualiza el registro R3, que es un operando fuente de I2. El término *antidependencia* se usa porque la restricción es similar a la de la dependencia verdadera, pero a la inversa: en lugar de que la primera instrucción produzca un valor que usa la segunda instrucción, la segunda instrucción destruye un valor que usa la primera instrucción.

RENOMBRAMIENTO DE REGISTROS

Hemos visto que permitir la emisión desordenada de instrucciones y/o la finalización desordenada puede dar origen a dependencias de salida y antidependencias. Estas dependencias son distintas de las dependencias de datos verdaderas y de los conflictos en los recursos, que reflejan el flujo de datos a través de un programa y la secuencia de ejecución. Las dependencias de salida y las antidependencias, por su parte, surgen porque los valores de los registros no pueden reflejar ya la secuencia de valores dictada por el flujo del programa.

Cuando las instrucciones se emiten y se completan secuencialmente, es posible especificar el contenido de cada registro en cada punto de la ejecución. Cuando se usan técnicas de desordenación, los valores de los registros no pueden conocerse completamente en cada instante temporal considerando solo la secuencia de instrucciones dictada por el programa. En realidad, los valores entran en conflicto por el uso de los registros, y el procesador debe resolver tales conflictos deteniendo ocasionalmente alguna etapa del cauce.

Las antidependencias y las dependencias de salida son dos ejemplos de conflictos de almacenamiento. Varias instrucciones compiten por el uso de los mismos registros, generando restricciones en el cauce que reducen las prestaciones. El problema se agudiza cuando se utilizan técnicas de optimización de registros (estudiadas en el Capítulo 13), ya que estas técnicas del compilador intentan maximizar el uso de registros, maximizando por tanto el número de conflictos de almacenamiento.

Un método para hacer frente a este tipo de conflictos de almacenamiento se basa en una solución tradicional para los conflictos en los recursos: la duplicación de recursos. En este contexto, la técnica

se conoce como **renombramiento de registros**. Básicamente, el hardware del procesador asigna dinámicamente los registros, que están asociados con los valores que necesitan las instrucciones en diversos instantes de tiempo. Cuando se crea un nuevo valor de registro (es decir, cuando se ejecuta una instrucción que tiene un registro como operando destino) se asigna un nuevo registro para ese valor. Las instrucciones posteriores que accedan a ese valor como operando fuente en ese registro tienen que sufrir un proceso de renombramiento: las referencias a registros de esas instrucciones han de revisarse para referenciar el registro que contiene el valor que se necesita. De este modo, las referencias a un mismo registro original en diferentes instrucciones pueden referirse a distintos registros reales, suponiendo diferentes valores.

Consideremos cómo se podría usar el renombramiento de registros en el fragmento de código que estamos examinando:

$$\begin{aligned}
&\texttt{I1:}\quad R3_b \leftarrow R3_a \;\texttt{op}\; R5_a \\
&\texttt{I2:}\quad R4_b \leftarrow R3_b + 1 \\
&\texttt{I3:}\quad R3_c \leftarrow R5_a + 1 \\
&\texttt{I4:}\quad R7_b \leftarrow R3_c \;\texttt{op}\; R4_b
\end{aligned}$$

La referencia a un registro sin el subíndice alude a una referencia a un registro lógico encontrada en la instrucción. La referencia a un registro con el subíndice alude a un registro hardware asignado para contener un nuevo valor. Cuando se hace una nueva asignación para un registro lógico particular, se hace que las referencias de instrucciones posteriores a ese registro lógico como operando fuente se refieran al registro hardware asignado más recientemente (reciente en términos de la secuencia de instrucciones del programa).

En este ejemplo, la creación del registro $R3_c$ en la instrucción I3 evita la antidependencia de la segunda instrucción y la dependencia de salida de la primera instrucción, y no impide que I4 acceda a un valor correcto. El resultado es que I3 puede emitirse inmediatamente; sin renombramiento, I3 no puede emitirse hasta que la primera instrucción haya finalizado y la segunda instrucción se haya emitido.

PARALELISMO DE LA MÁQUINA

En los párrafos precedentes hemos estudiado tres técnicas hardware que se pueden usar en un procesador superescalar para aumentar las prestaciones: duplicación de recursos, emisión desordenada y renombramiento. En [SMIT89] se presentó un estudio que aclara la relación entre estas técnicas. El estudio utilizó una simulación que modelaba una máquina de las características del MIPS R2000, aumentada con algunas características superescalares. Se simularon diferentes secuencias de programa.

La Figura 14.5 muestra los resultados. En las dos gráficas, el eje vertical corresponde al incremento de velocidad medio de la máquina superescalar respecto a la máquina escalar. El eje horizontal muestra los resultados para cuatro organizaciones del procesador alternativas. La máquina base no duplica ninguna de las unidades funcionales, pero puede emitir instrucciones desordenadamente. La segunda configuración duplica la unidad funcional de carga/almacenamiento que accede a la caché de datos. La tercera configuración duplica la ALU, y la cuarta configuración duplica tanto la unidad de carga/almacenamiento como la ALU. En las dos gráficas, los resultados se muestran para tamaños de ventana de

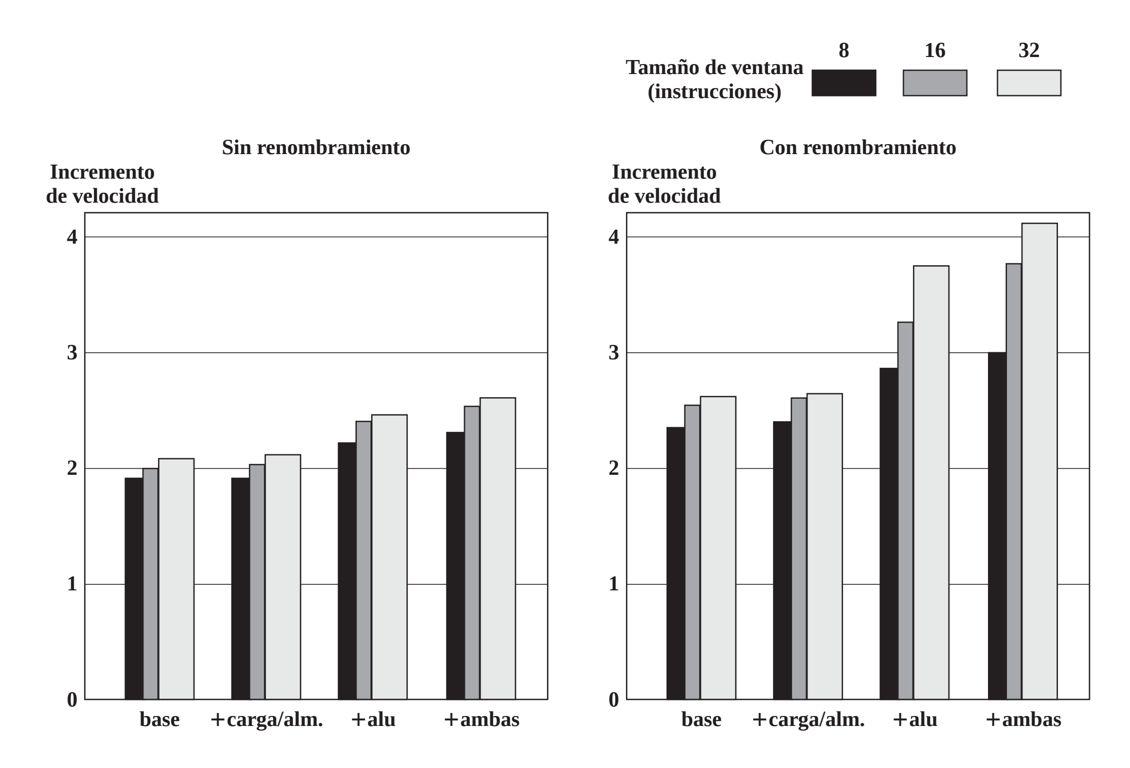


Figura 14.5. Incremento de velocidad de diversas organizaciones de una máquina, sin dependencias relativas al procedimiento.

instrucciones de 8, 16 y 32 instrucciones, que determinan el grado de anticipación que puede tener el procesador. La diferencia entre las dos gráficas es que en la segunda se permite el renombramiento de registros. Esto es equivalente a decir que la primera gráfica refleja una máquina limitada por todas las dependencias, mientras que la segunda gráfica corresponde a una máquina limitada solo por las dependencias verdaderas.

Las dos gráficas, combinadas, ofrecen algunas conclusiones importantes. La primera es que probablemente no merece la pena añadir unidades funcionales sin renombramiento de registros. Hay algunas mejoras de poca importancia en las prestaciones, pero con el coste de una complejidad del hardware aumentada. Con el renombramiento de registros, que elimina las antidependencias y las dependencias de salida, se logran ganancias notables añadiendo más unidades funcionales. Observe, no obstante, que hay una diferencia significativa en la ganancia alcanzable con el uso de una ventana de ocho instrucciones frente a la que se consigue usando una ventana de instrucciones mayor. Esto indica que si la ventana de instrucciones es demasiado pequeña, las dependencias de datos impiden la utilización efectiva de las unidades funcionales adicionales; es importante que el procesador pueda mirar hacia delante bastante lejos en busca de instrucciones independientes que permitan aprovechar más el hardware.

PREDICCIÓN DE SALTOS

Cualquier máquina segmentada de altas prestaciones debe estudiar la cuestión del tratamiento de los saltos. Por ejemplo, el Intel 80486 soluciona el problema captando tanto la siguiente instrucción

secuencial a la de bifurcación como la instrucción destino del salto. Sin embargo, como hay dos etapas en el cauce entre la precaptación y la ejecución, esta estrategia incurre en un retardo de dos ciclos cuando se produce el salto.

Con la llegada de las máquinas RISC, se exploró la estrategia de salto retardado. Esta permite al procesador calcular el resultado de las instrucciones de salto condicional antes de que se precapten instrucciones innecesarias. Con este método, el procesador siempre ejecuta la instrucción que sigue inmediatamente a la de salto. Esto mantiene lleno el cauce mientras el procesador capta una nueva secuencia de instrucciones.

Con el desarrollo de las máquinas superescalares, la estrategia de salto retardado ha perdido interés. El motivo es que hay que ejecutar múltiples instrucciones en el ciclo de retardo, lo que plantea varios problemas relacionados con las dependencias entre instrucciones. Por ello, las máquinas superescalares han regresado a las técnicas de predicción de saltos anteriores a las de los RISC. Algunas, como el PowerPC 601, usan una técnica sencilla de predicción de saltos estática. Los procesadores más sofisticados, como el PowerPC 620 y el Pentium 4, usan predicción dinámica de saltos basada en el análisis de la historia de los saltos.

EJECUCIÓN SUPERESCALAR

Estamos ahora en condiciones de dar una visión de conjunto de la ejecución superescalar de programas, ilustrada en la Figura 14.6. El programa que se va a ejecutar consiste en una secuencia lineal de instrucciones. Se trata del programa estático tal como fue escrito por el programador o generado por el compilador. El proceso de captación de instrucciones, que incluye la predicción de saltos, se usa para formar un flujo dinámico de instrucciones. Se examinan las dependencias de este flujo, y el procesador puede eliminar las que sean artificiales. El procesador envía entonces las instrucciones a una ventana de ejecución. En esta ventana, las instrucciones ya no forman un flujo secuencial sino que están estructuradas de acuerdo a sus dependencias de datos verdaderas. El procesador lleva a cabo la

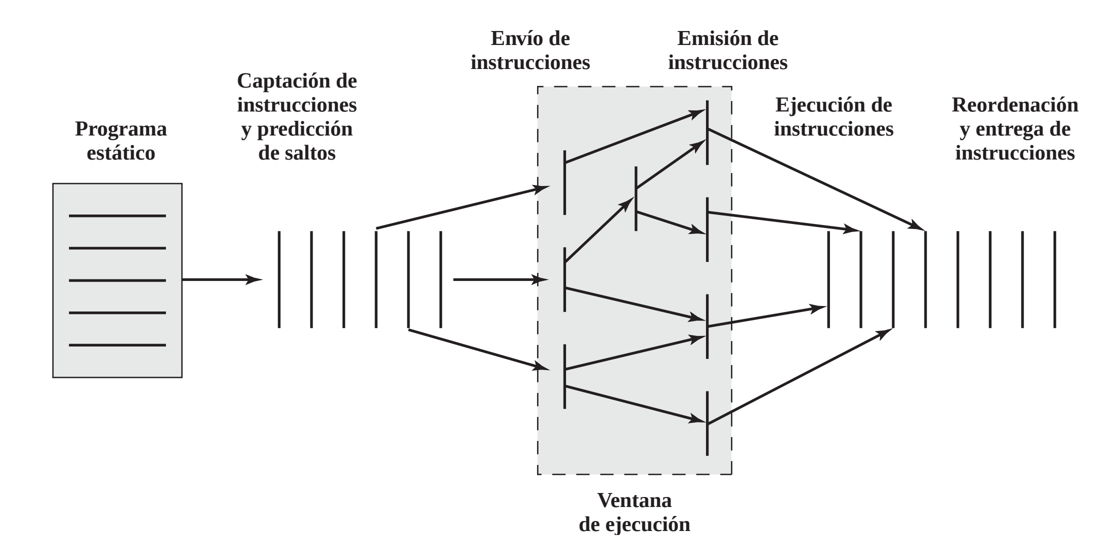


Figura 14.6. Representación conceptual del procesamiento superescalar [SMIT95].

etapa de ejecución de cada instrucción en un orden determinado por las dependencias de datos verdaderas y la disponibilidad de los recursos hardware. Por último, las instrucciones se vuelven a poner conceptualmente en un orden secuencial y sus resultados se almacenan.

Al último paso mencionado en el párrafo precedente se le llama **entregar (*commit*)**, o **retirar (*retire*)**, la instrucción. Este paso es necesario por la siguiente razón. Debido al uso de múltiples cauces paralelos, las instrucciones pueden terminar en un orden diferente al que muestran en el programa estático. Además, la utilización de predicción de saltos y ejecución especulativa significa que algunas instrucciones pueden completar su ejecución y después ser desechadas porque el salto que llevaba a ellas no se produjo. Por consiguiente, el almacenamiento permanente y los registros visibles por el programa no se pueden actualizar inmediatamente después de que las instrucciones finalicen su ejecución. Los resultados han de mantenerse en algún tipo de almacenamiento temporal que sea utilizable por instrucciones dependientes y después convertido en permanente una vez que se determine que el modelo secuencial habría ejecutado la instrucción.

IMPLEMENTACIÓN SUPERESCALAR

Basándonos en lo discutido hasta ahora, podemos hacer algunos comentarios generales sobre el hardware que requiere el procesador en la aproximación superescalar. En [SMIT95] se enumeran los siguientes elementos principales:

- Estrategias de captación de instrucciones que capten simultáneamente múltiples instrucciones, a menudo prediciendo los resultados de las instrucciones de salto condicional y captando más allá de ellas. Estas funciones requieren la utilización de múltiples etapas de captación y decodificación, y lógica de predicción de saltos.
- Lógica para determinar dependencias verdaderas entre valores de registros, y mecanismos para comunicar esos valores a donde sean necesarios durante la ejecución.
- Mecanismos para iniciar o emitir múltiples instrucciones en paralelo.
- Recursos para la ejecución en paralelo de múltiples instrucciones, que incluyan múltiples unidades funcionales segmentadas y jerarquías de memoria capaces de atender múltiples referencias a memoria.
- Mecanismos para entregar el estado del procesador en el orden correcto.

14.3. PENTIUM 4

Aunque el concepto de diseño superescalar se asocia generalmente con la arquitectura RISC, se pueden aplicar los mismos principios superescalares a una máquina CISC. El ejemplo más notable de ello tal vez sea el Pentium. Es interesante observar la evolución de los conceptos superescalares en la línea Intel. El 80486 era claramente una máquina CISC tradicional, sin elementos superescalares. El Pentium original tenía una modesta componente superescalar, que consistía en la utilización de dos unidades de ejecución de enteros independientes. El Pentium Pro introdujo un diseño completamente superescalar. Los siguientes modelos de Pentium han refinado y mejorado el diseño superescalar.

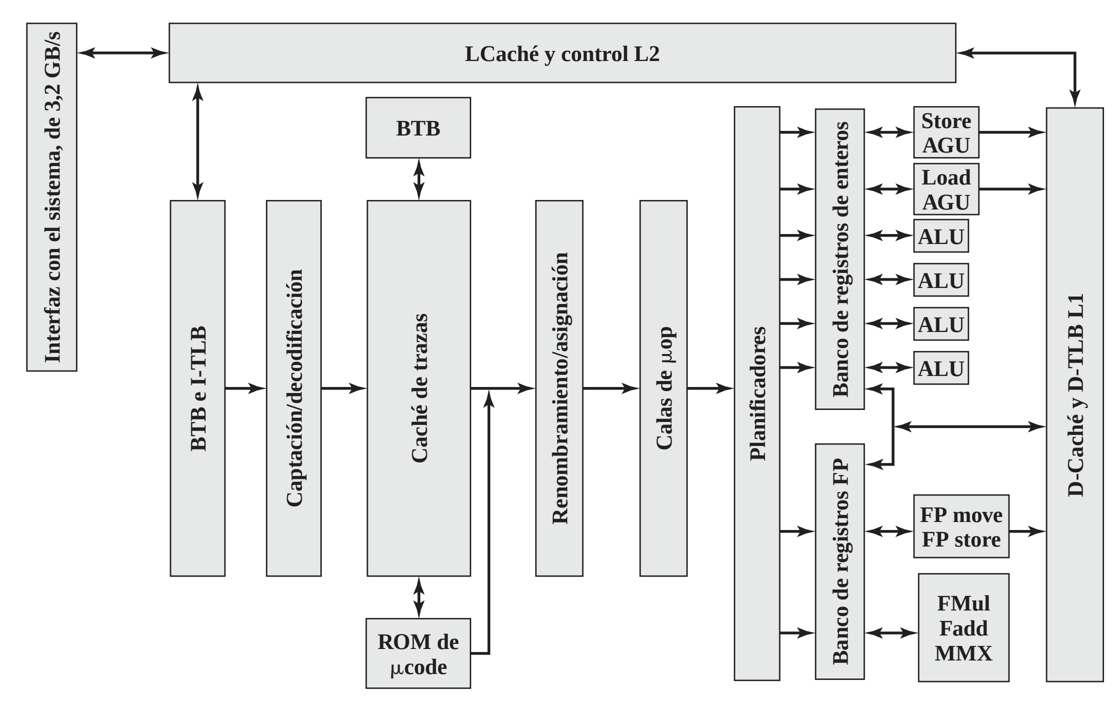


Figura 14.7. Diagrama de bloques del Pentium 4.

En la Figura 4.13 se mostró un diagrama de bloques general del Pentium 4. La Figura 14.7 representa la misma estructura de una manera más adecuada para la discusión sobre segmentación de esta sección. El funcionamiento del Pentium 4 se puede resumir como sigue:

1. El procesador capta instrucciones de memoria en el orden en que aparecen en el programa estático.
2. Cada instrucción se traduce en una o más instrucciones RISC de tamaño fijo conocidas como **microoperaciones** o **micro-ops**.
3. El procesador ejecuta las micro-ops en una organización de cauce superescalar, de modo se pueden ejecutar desordenadas.
4. El procesador entrega los resultados de la ejecución de cada micro-op al conjunto de registros, en el orden del flujo del programa original.

En realidad, la arquitectura del Pentium 4 consta de una envoltura CISC con un núcleo RISC interno. Las micro-ops RISC internas pasan a través de un cauce con al menos veinte etapas (Figura 14.8); en algunos casos, las micro-ops necesitan múltiples etapas de ejecución, lo que se traduce en un cauce aún más largo. Esto contrasta con el cauce de cinco etapas (Figura 12.19) utilizado en los procesadores Intel x86 y en el Pentium.

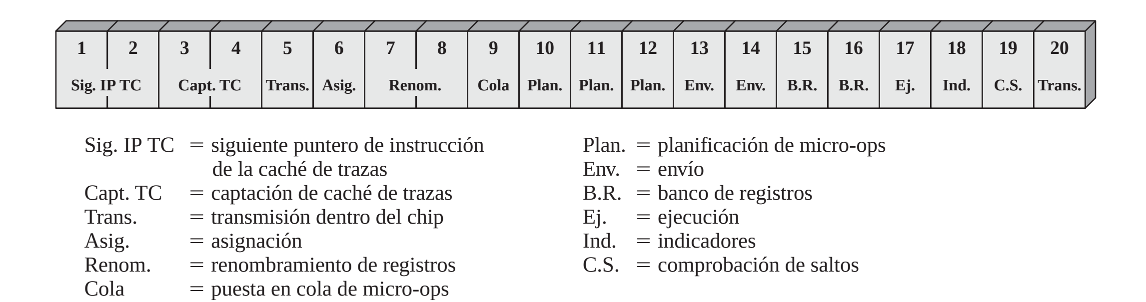


Figura 14.8. Cauce segmentado del Pentium 4.

A continuación se expone el funcionamiento del cauce del Pentium 4, usando la Figura 14.9 como referencia.

INTERFAZ EXTERNA

Generación de micro-ops. La organización del Pentium 4 incluye una interfaz «en orden» (Figura 14.9a) que puede considerarse fuera del ámbito del cauce segmentado representado en la Figura 14.8. Esta interfaz termina en una caché de instrucciones L1, llamada caché de trazas, que es donde comienza realmente el cauce segmentado. Habitualmente, el procesador opera a partir de la caché de trazas; cuando ocurre un fallo de dicha caché, la interfaz externa «en orden» introduce nuevas instrucciones en ella.

Con la ayuda del buffer de destino de saltos y el buffer de traducción anticipada de instrucciones (BTB e I-TLB), la unidad de captación/decodificación capta instrucciones máquina del Pentium 4 desde la caché L2 (64 bytes cada vez). Normalmente, las instrucciones se captan secuencialmente, de tal modo que cada captación de una línea de caché L2 incluye la siguiente instrucción a captar. La predicción de saltos realizada por el BTB y el I-TLB puede alterar esta operación de captación secuencial. El I-TLB traduce la dirección lineal del puntero de instrucciones a una dirección física necesaria para acceder a la caché L2. Para determinar qué instrucciones deben captarse a continuación se usa predicción de saltos estática en el BTB.

Una vez que las instrucciones se captan, la unidad de captación/decodificación comprueba los bytes para determinar los límites de las instrucciones; se trata de una operación necesaria debida a la longitud variable de las instrucciones del Pentium. El decodificador traduce cada instrucción máquina en varias micro-ops (de una hasta cuatro), cada una de las cuales es una instrucción RISC de 118 bits. Observe que en comparación las máquinas RISC más puras tienen una longitud de instrucción de solo 32 bits. La longitud mayor de las micro-ops es necesaria para adaptarse a las operación más complejas del Pentium. Sin embargo, las micro-ops son más fáciles de manejar que las instrucciones originales de las cuales derivan.

Puntero de instrucción siguiente de la caché de trazas. Las dos primeras etapas del cauce (Figura 14.9b) se ocupan de la selección de instrucciones en la caché de trazas e incluyen un mecanismo de predicción de saltos separado del que se ha descrito en la sección precedente. El Pentium 4 usa una estrategia de predicción dinámica de saltos basada en la historias de las ejecuciones recientes de las instrucciones de salto. Se utiliza un buffer de destino de saltos (*branch target buffer*, BTB) que

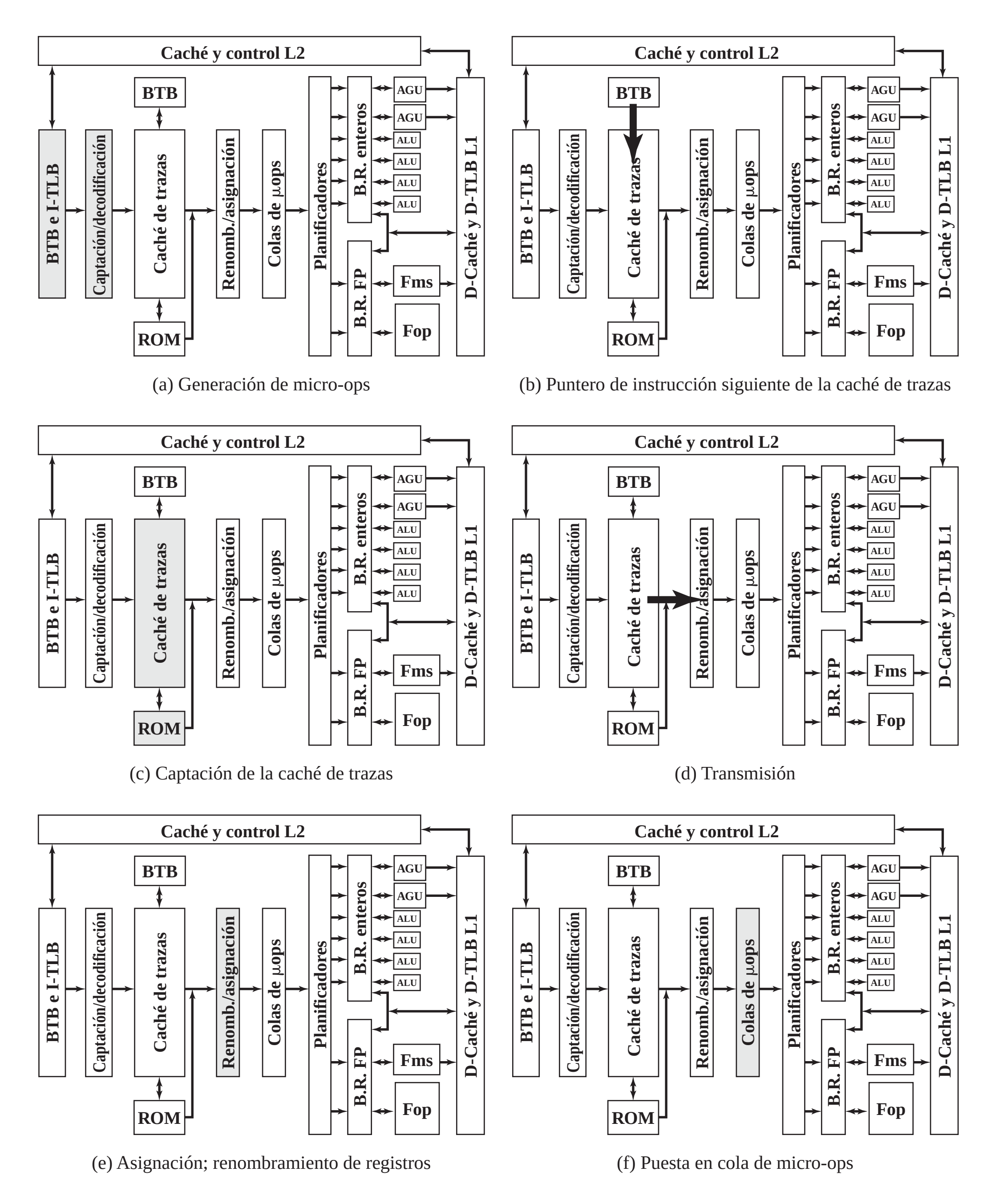


Figura 14.9. Funcionamiento del cauce segmentado del Pentium 4 *(continúa)*.

guarda información sobre las instrucciones de salto encontradas recientemente. Siempre que aparece una instrucción de salto en el flujo de instrucciones, se comprueba el BTB. Si ya existe un elemento en el BTB, la unidad de instrucciones se guía por la información de historia guardada en ese elemento para determinar si predice que se producirá el salto. Si se predice un salto, la dirección destino del salto asociada con este elemento se utiliza para precaptar la instrucción destino del salto.

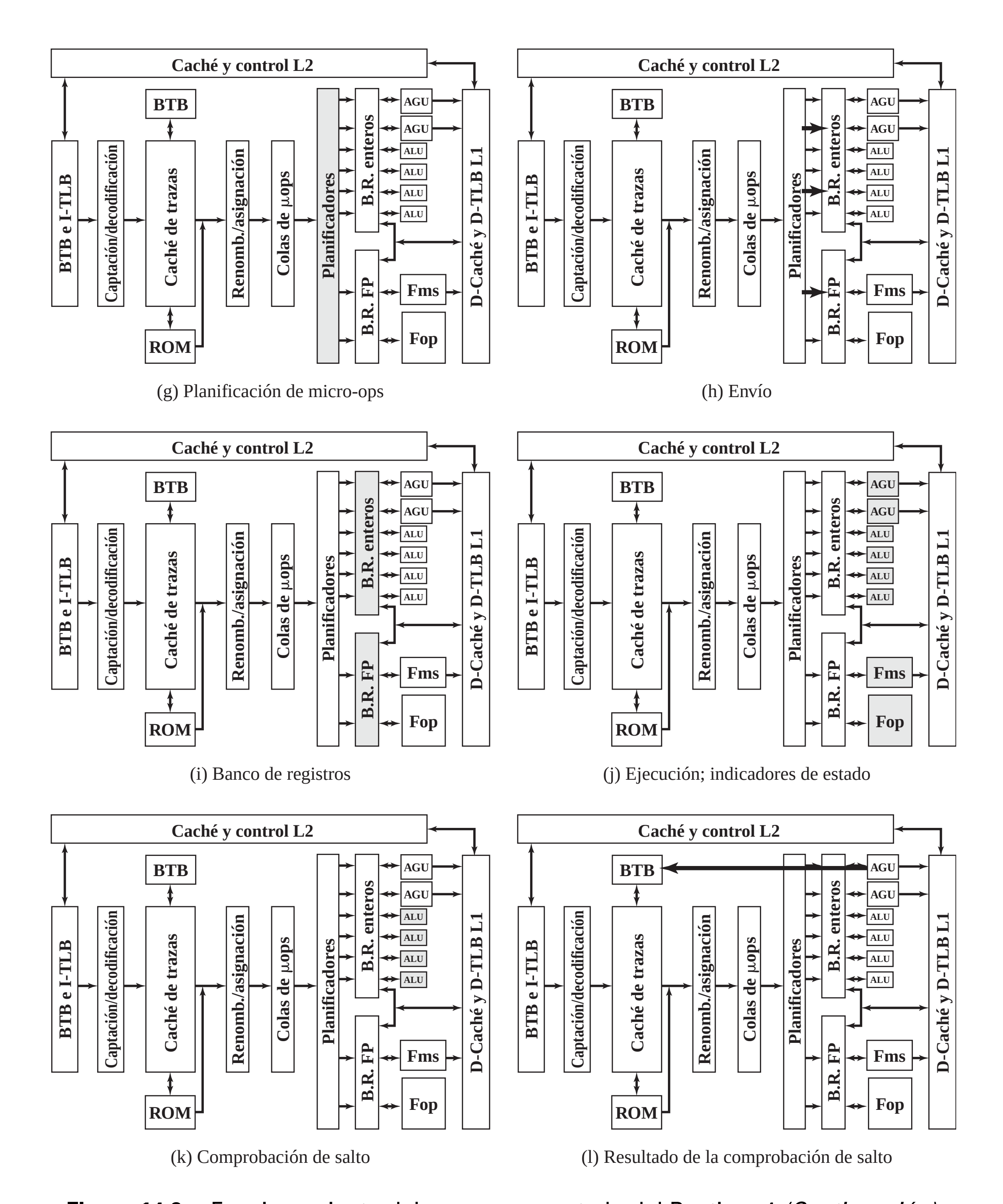


Figura 14.9. Funcionamiento del cauce segmentado del Pentium 4 *(Continuación)*.

Una vez que se ejecuta la instrucción, la parte de historia del elemento adecuado se actualiza para que refleje el resultado de la instrucción de salto. Si la instrucción no está representada en el BTB, se carga su dirección en una entrada del BTB; si es preciso, se borra un elemento más antiguo.

La descripción de los dos párrafos precedentes se ajusta, en términos generales, a la estrategia de predicción de saltos que se utiliza en el Pentium original, así como en posteriores modelos de

Pentium, incluyendo el Pentium 4. Sin embargo, en el caso del Pentium, se usa un esquema de historia de dos bits relativamente sencillo. Los modelos de Pentium posteriores tienen cauces mucho más largos (20 etapas en el Pentium 4 comparadas con las cinco del Pentium) y por tanto la penalización debida a un error de predicción es mayor. Así pues, los últimos modelos de Pentium emplean un esquema de predicción de saltos más complejo con más bits de historia para reducir la tasa de fallos de predicción.

El BTB del Pentium 4 está organizado como una caché asociativa por conjuntos de cuatro vías con 512 líneas. Cada elemento utiliza la dirección de la instrucción de salto como etiqueta. El elemento también incluye la dirección destino del salto de la última vez que se produjo y un campo de historia de cuatro bits. El empleo de cuatro bits de historia contrasta con los dos bits usados en el Pentium original y en la mayoría de los procesadores superescalares. Con cuatro bits, el mecanismo del Pentium 4 puede tener en cuenta una historia más larga para predecir saltos. El algoritmo usado se conoce como algoritmo de Yeh [YEH91]. Los autores de este algoritmo han demostrado que proporciona una reducción significativa de los fallos de predicción comparado con los algoritmos que utilizan solo dos bits de historia [EVER98].

Los saltos condicionales que no tienen historia en el BTB se predicen usando un algoritmo de predicción estática, de acuerdo con las siguientes reglas:

- En las instrucciones de salto que no son relativas a IP, se predice que se producirá el salto si se trata de una instrucción de retorno, y que no se saltará en cualquier otro caso.
- En los saltos condicionales hacia atrás relativos a IP, se predice que se producirá el salto. Esta regla refleja el comportamiento típico en los bucles.
- En los saltos condicionales hacia delante, se predice que no se producirá el salto.

Captación de la caché de trazas. La caché de trazas (Figura 14.9c) toma las micro-ops ya decodificadas del decodificador de instrucciones y las ensambla en secuencias ordenadas según el programa conocidas como trazas. Las micro-ops se captan secuencialmente de la caché de trazas, dependiendo de la lógica de predicción de saltos.

Unas cuantas instrucciones necesitan más de cuatro micro-ops. Estas instrucciones se transfieren a la ROM de microcódigo, que contiene las series de micro-ops (cinco o más) asociadas a cada instrucción máquina compleja. Por ejemplo, una instrucción de cadena puede traducirse en una secuencia repetitiva y muy grande de micro-ops (incluso de cientos de ellas). Por consiguiente, la ROM de microcódigo es una unidad de control microprogramada en el sentido que se discute en la Parte Cuatro. Después de que la ROM de microcódigo termina de generar la secuencia de micro-ops de la instrucción actual del Pentium, se reanuda la búsqueda de instrucciones desde la caché de trazas.

Transmisión. La quinta etapa (Figura 14.9d) del cauce del Pentium 4 entrega instrucciones decodificadas desde la caché de trazas al módulo de renombramiento/asignación.

LÓGICA DE EJECUCIÓN DESORDENADA

Esta parte del procesador reordena las micro-ops para permitir que se ejecuten tan pronto como sus operandos de entrada esté listos.

Asignación. La etapa de asignación (Figura 14.9e) asigna los recursos necesarios para la ejecución. Lleva a cabo las siguientes funciones:

- Si un recurso necesario, tal como un registro, no está disponible para alguna de las tres micro-ops que llegan a esta etapa de asignación durante un ciclo de reloj, detiene el cauce.
- Asigna un elemento del buffer de reordenación (*reorder buffer,* ROB), que sigue la pista del estado de finalización de una de las 126 micro-ops que pueden estar en proceso en cualquier momento.
- Asigna uno de los 128 registros de enteros o de coma flotante al dato resultante de la micro-op, y posiblemente un buffer de carga o almacenamiento usado para seguir la pista a una de las 48 cargas o 24 almacenamientos en el cauce de la máquina.
- Asigna un elemento en una de las dos colas de micro-ops situadas frente a los planificadores de instrucciones.

El ROB es un buffer circular que puede contener hasta 126 micro-ops y también contiene los 128 registros hardware. Cada elemento del buffer consta de los siguientes campos:

- **Estado:** indica si la micro-op está lista para ejecutarse, ha sido enviada a su ejecución o ha terminado de ejecutarse y está lista para su retiro.
- **Dirección de memoria:** dirección de la instrucción del Pentium que generó la micro-op.
- **Micro-op:** la operación propiamente dicha.
- **Registro alias:** si la micro-op referencia uno de los 16 registros de la arquitectura, este campo redirecciona esa referencia a uno de los 128 registros hardware.

Las micro-ops entran al ROB en orden. Después son enviadas desde el ROB a la unidad de envío/ejecución sin orden. El criterio para que una micro-op sea enviada es que la unidad de ejecución apropiada y todos los datos que requiera la micro-op se encuentren disponibles. Por último, las micro-ops se retiran del ROB en orden. Para lograr esto, las micro-ops se retiran empezando por la más antigua después de que cada micro-op se haya señalado como lista para ser retirada.

Renombramiento de registros. La etapa de renombramiento (Figura 14.9e) transforma las referencias a los 16 registros de la arquitectura (ocho registros de coma flotante, más EAX, EBX, ECX, EDX, ESI, EDI, EBP y ESP) a un conjunto de 128 registros físicos. La etapa elimina dependencias falsas causadas por el limitado número de registros de la arquitectura, preservando las dependencias de datos verdaderas (lecturas después de escrituras).

Puesta en cola de micro-ops. Tras la asignación de recursos y el renombramiento de registros, las micro-ops se colocan en una de las dos colas de micro-ops (Figura 14.9f), donde permanecen hasta que haya sitio en los planificadores. Una de las colas se usa para operaciones de memoria (cargas y almacenamientos) y la otra para las micro-ops que no incluyen referencias a memoria. Cada cola sigue una disciplina FIFO (*First-In-First-Out*, primero en entrar, primero en salir), pero no hay ningún orden entre las colas. Es decir, cada micro-op puede leerse de una cola sin orden con respecto a las micro-ops de la otra cola. Esto proporciona mayor flexibilidad a los planificadores.

Planificación y envío de micro-ops. Los planificadores (Figura 14.9g) son responsables de recoger las micro-ops de las colas de micro-ops y enviarlas para su ejecución. Cada planificador busca micro-ops cuyo estado indique que tienen disponibles todos sus operandos. Si la unidad de ejecución precisada por una de esas micro-ops está disponible, el planificador capta la micro-op y la envía a la unidad de ejecución apropiada (Figura 14.9h). En un ciclo se pueden enviar hasta seis micro-ops. Si hay más de una micro-op disponible para una unidad de ejecución dada, el planificador las envía secuencialmente desde la cola. Este es un tipo de disciplina FIFO que favorece la ejecución en orden, pero en ese momento el flujo de instrucciones ha sido reorganizado tanto por las dependencias y los saltos que realmente está sin orden.

Hay cuatro puertos que conectan los planificadores a las unidades de ejecución. El Puerto 0 se usa tanto para instrucciones de enteros como de coma flotante, con la excepción de las operaciones con enteros sencillas y la gestión de las predicciones de salto erróneas, que están asignadas al Puerto 1. Además, las unidades de ejecución MMX también están asignadas a estos dos puertos. Los puertos restantes se utilizan para cargas y almacenamientos en memoria.

UNIDADES DE EJECUCIÓN DE ENTEROS Y DE COMA FLOTANTE

Los bancos de registros de enteros y de coma flotante son la fuente de las operaciones pendientes en las unidades de ejecución (Figura 14.9i). Las unidades de ejecución toman valores tanto de los bancos de registros como de la caché de datos L1 (Figura 14.9j). Una etapa del cauce separada se usa para calcular los indicadores de estado (por ejemplo, cero, negativo), que son normalmente la entrada de las instrucciones de salto.

Una etapa del cauce posterior realiza la comprobación de saltos (Figura 14.9k). Esta función compara el resultado real del salto con la predicción. Si una predicción de salto resulta fallida, habrá micro-operaciones en varias etapas de procesamiento que tengan que ser retiradas del cauce. El destino correcto del salto se proporciona entonces a la unidad de predicción de saltos durante una etapa de transmisión (Figura 14.9l), que reinicia el cauce completo desde la nueva dirección de destino.

14.4. POWERPC

La arquitectura del PowerPC desciende directamente del IBM 801, el RT PC, y el RS/6000, al que también se alude como una implementación de la arquitectura POWER. Todas estas son máquinas RISC, pero la primera de la serie que presentó características superescalares fue el RS/6000. La primera implementación de la arquitectura PowerPC, el 601, tiene un diseño superescalar bastante similar al del RS/6000. Los siguientes modelos del PowerPC llevan más lejos el concepto de arquitectura superescalar. En esta sección nos centraremos en el 601, que proporciona un buen ejemplo de diseño superescalar basado en un RISC. Al final de la sección, consideraremos brevemente el 620.

POWERPC 601

La Figura 14.10 ofrece una visión general de la organización del 601. Como otras máquinas superescalares, el 601 se divide en unidades funcionales independientes para permitir la ejecución en paralelo.

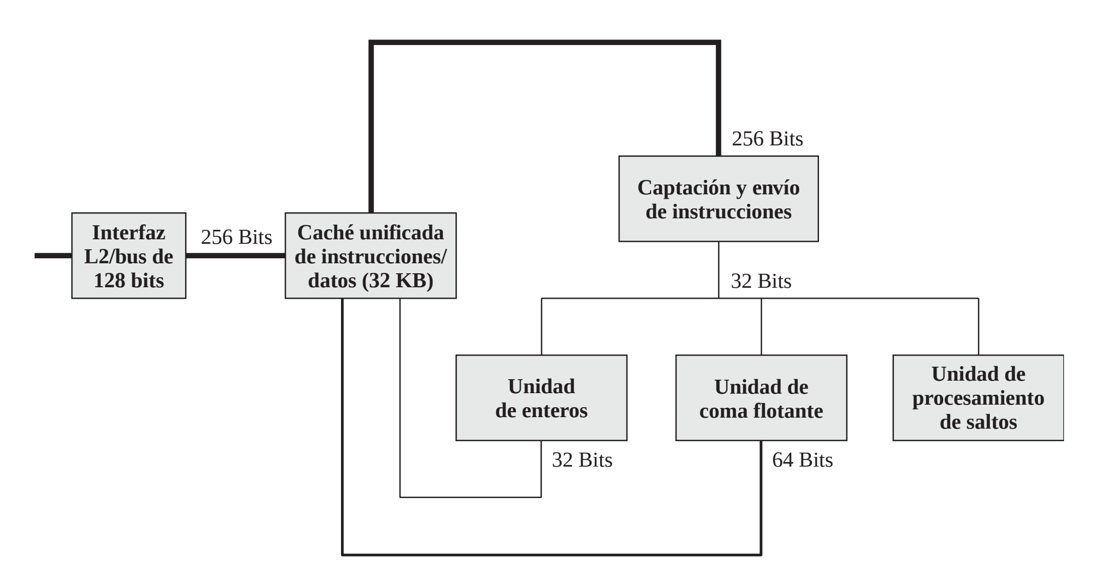


Figura 14.10. Diagrama de bloques del PowerPC 601.

Concretamente, el núcleo del 601 consta de tres unidades de ejecución segmentadas independientes para enteros, coma flotante y procesamiento de saltos. Juntas, estas unidades pueden ejecutar tres instrucciones al mismo tiempo, ofreciendo un diseño superescalar de grado 3.

La Figura 14.11 muestra un esquema lógico de la arquitectura del 601, destacando el flujo de instrucciones entre las unidades funcionales. La unidad de captación puede precaptar de la caché hasta ocho instrucciones al mismo tiempo. La unidad de caché soporta una caché combinada de instrucciones/datos y es responsable de suministrar instrucciones a las otras unidades y datos a los registros. La lógica de arbitraje de la caché envía a ésta la dirección del acceso de mayor prioridad.

Unidad de envío. La unidad de envío toma instrucciones de la caché y las carga en una cola de envío, que puede contener ocho instrucciones a la vez. Procesa esta secuencia de instrucciones para suministrar un flujo constante de instrucciones a las unidades de procesamiento de saltos, de enteros, y de coma flotante. La mitad superior de la cola actúa sencillamente como un buffer que contiene instrucciones hasta que estas se transfieren a la mitad inferior. Su misión es asegurar que la unidad de envío no se retrasa esperando instrucciones provenientes de la caché. En la mitad inferior, las instrucciones se envían según el siguiente esquema:

- **Unidad de procesamiento de saltos:** se encarga de todas las instrucciones de salto. La instrucción inferior de este tipo en la mitad de abajo de la cola de envío se emite a la unidad de procesamiento de saltos si esta puede aceptarla.
- **Unidad de coma flotante:** se ocupa de todas las instrucciones de coma flotante. La instrucción inferior de este tipo en la mitad de abajo de la cola de envío se emite a la unidad de coma flotante si el cauce de instrucciones de esta unidad no está lleno.
- **Unidad de enteros:** se encarga de las instrucciones de enteros, las de carga/almacenamiento entre el banco de registros y la caché, y las instrucciones de comparación de enteros. Una instrucción de enteros se emite solo después de que se haya filtrado hasta el extremo inferior de la cola de envío.

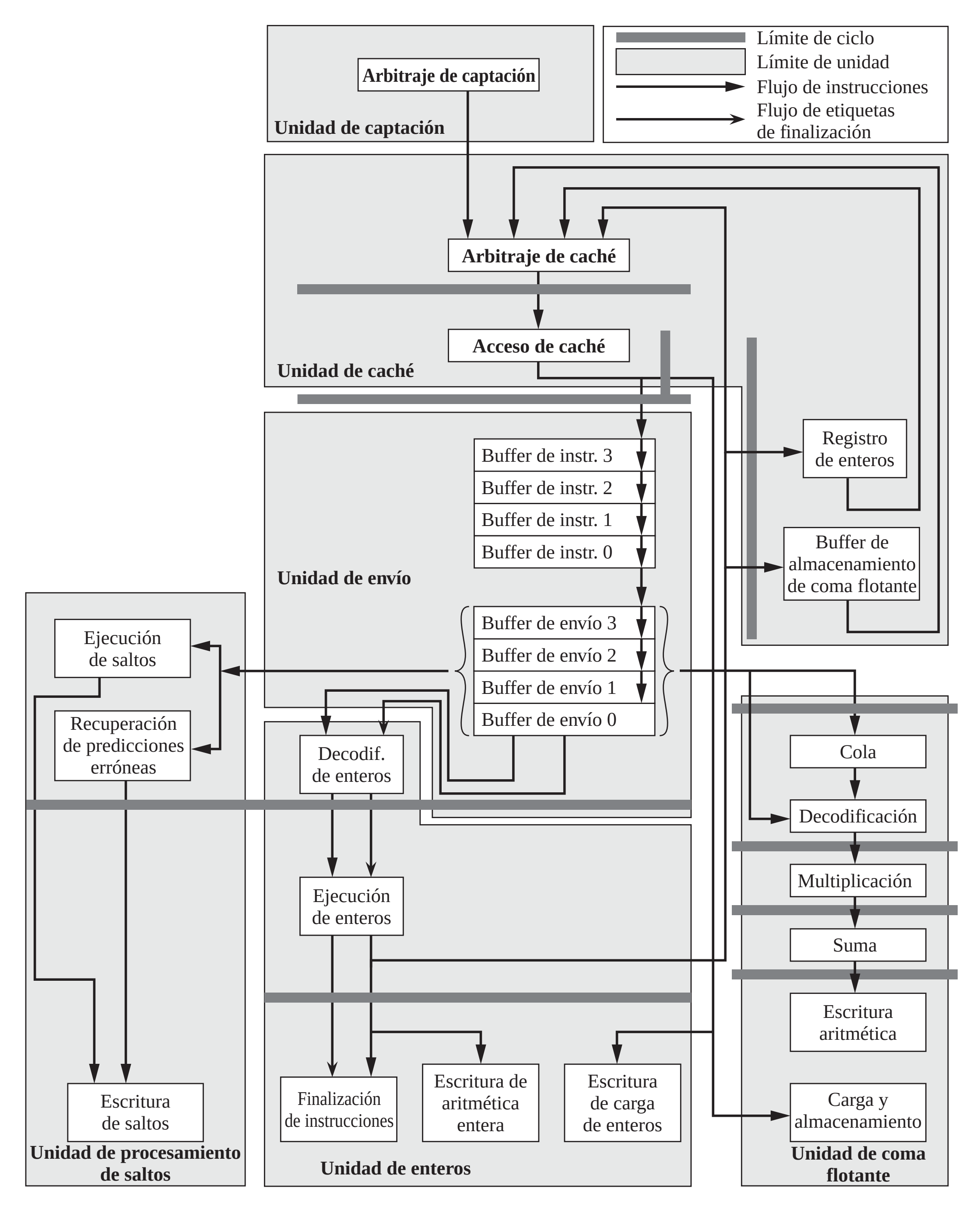


Figura 14.11. Estructura de cauces del PowerPC 601 [POTT94].

La posibilidad de emitir instrucciones de salto y de coma flotante desordenadamente desde la cola de envío ayuda a mantener llenos los cauces de instrucciones de las unidades de procesamiento de saltos y de coma flotante, y a mover instrucciones a través de la cola de envío lo más rápido posible.

La unidad de envío también contiene lógica que le permite calcular la dirección de precaptación. Continúa captando instrucciones secuencialmente hasta que una instrucción de salto se transfiere a la mitad inferior de la cola de envío. Cuando la unidad de procesamiento de saltos procesa una instrucción, puede actualizar la dirección de precaptación de manera que las siguientes instrucciones se capten de la nueva dirección y entren en la cola de envío.

Cauces de instrucciones. La Figura 14.12 ilustra los cauces de instrucciones de las distintas unidades. Hay un ciclo de captación común a todas las instrucciones; este tiene lugar antes de que una instrucción se envíe a una unidad concreta. El segundo ciclo comienza con el envío de una instrucción a una unidad particular. Este se solapa con otras actividades dentro de la unidad. En cada ciclo de reloj, la unidad de envío examina los cuatro elementos inferiores de la cola de instrucciones y envía hasta tres instrucciones.

En las instrucciones de salto, el segundo ciclo supone la decodificación y ejecución de instrucciones además de la predicción de saltos. Esta última actividad se estudia en la siguiente subsección.

La unidad de enteros se ocupa de las instrucciones que causan una operación de carga/almacenamiento de/en memoria (incluyendo carga/almacenamiento de coma flotante), una transferencia registro a registro, o una operación de la ALU. En el caso de carga/almacenamiento, hay un ciclo de generación de dirección seguido del envío de la dirección resultante a la caché y, si es necesario, un ciclo de escritura. En otras instrucciones, la caché no está involucrada y hay un ciclo de ejecución seguido de una escritura en un registro.

Las instrucciones de coma flotante siguen un cauce similar, pero con dos ciclos de ejecución, que reflejan la complejidad de las operaciones de coma flotante.

Hay otros puntos que son dignos de atención. El registro de condición contiene ocho campos de código de condición de cuatro bits independientes. Esto permite guardar múltiples códigos de condición,

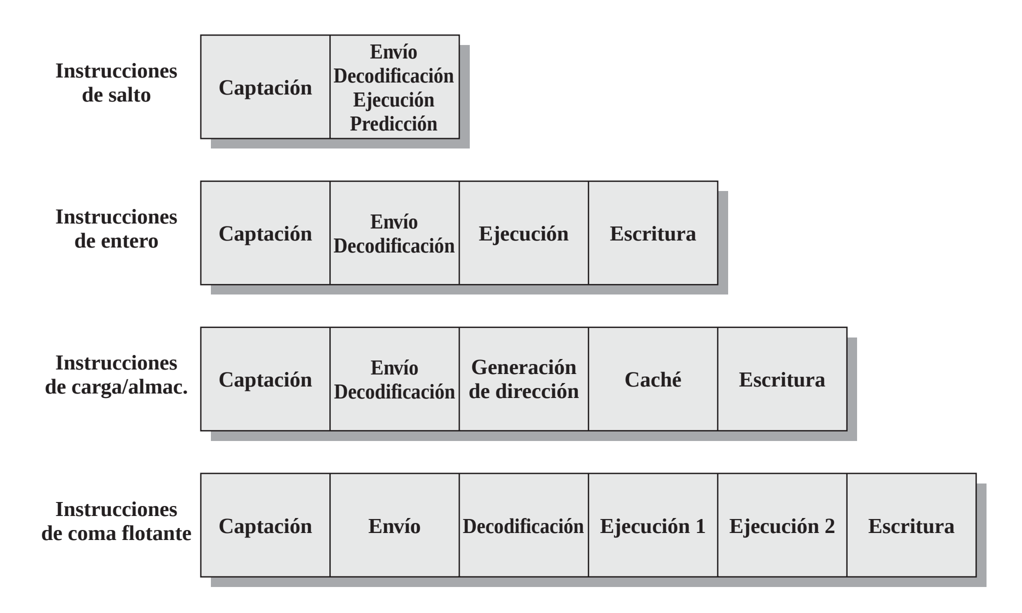


Figura 14.12. Cauce segmentado del PowerPC 601.

lo que reduce el interbloqueo o dependencia entre instrucciones. Por ejemplo, el compilador puede transformar la secuencia

comparar
saltar
comparar
saltar
•
•
•

en la secuencia

comparar
comparar
•
•
•
saltar
saltar
•
•
•

Como cada unidad funcional puede enviar sus códigos de condición a diferentes campos del registro de condición, se pueden evitar los interbloqueos entre instrucciones causados por el hecho de compartir los códigos de condición.

La presencia de los registros salvar y restaurar (*Save and Restore Registers”,* SRR) en el procesador de saltos le permite gestionar interrupciones sencillas e interrupciones software sin involucrar lógica de otras unidades funcionales. De este modo, los servicios sencillos del sistema operativo se pueden ejecutar rápidamente sin manipulación de estados o sincronización complicadas entre las unidades funcionales.

Como el 601 puede emitir instrucciones de salto y de coma flotante desordenadamente, se necesitan controles que aseguren la correcta ejecución. Cuando existe una dependencia (es decir, cuando una instrucción necesita un operando que aún no ha sido calculado por una instrucción previa), el cauce de la unidad correspondiente se detiene.

PROCESAMIENTO DE SALTOS

La clave de las altas prestaciones de una máquina RISC o superescalar es su habilidad para optimizar el uso del cauce. Típicamente, el elemento más crítico del diseño es cómo manejar los saltos. En el PowerPC, el procesamiento de saltos es responsabilidad de la unidad de saltos. La unidad está diseñada de manera tal que en muchos casos, los saltos no tengan efecto en el ritmo de ejecución de las otras unidades; a este tipo de saltos se les llama saltos de cero ciclos. Para conseguir saltos de cero ciclos se emplean las siguientes estrategias:

1. Se utiliza una lógica que examina el buffer de envío en busca de saltos. Se generan direcciones destino de salto cuando aparece un salto en la mitad inferior de la cola y no hay saltos anteriores pendientes de ejecución.
2. Se intenta determinar el resultado de los saltos condicionales. Si el código de condición se ha ajustado por adelantado lo suficientemente pronto, este resultado puede determinarse. En todo caso, tan pronto como se encuentre una instrucción de salto, la lógica determina si el salto:
   (a) Se producirá; este es el caso de los saltos incondicionales y de las bifurcaciones condicionales cuyo código de condición se conoce e indica que hay que saltar.
   (b) No se producirá; este es el caso de bifurcaciones condicionales cuyo código de condición se conoce e indica que no hay salto.
   (c) No se puede determinar todavía. En este caso, se supone que se producirá el salto en los saltos hacia atrás (típicos en los bucles) y se estima que no se producirá en saltos hacia delante. Las instrucciones secuenciales posteriores a la instrucción de salto se pasan a las unidades de ejecución de manera condicional. Una vez que el código de condición se ajusta en la unidad de ejecución, la unidad de salto o bien cancela las instrucciones que hay en el cauce y continúa por la instrucción destino captada, si el salto se produce; o bien indica que se ejecuten las instrucciones condicionales. El compilador puede usar un bit del código de la instrucción para invertir este comportamiento implícito.

La incorporación de una estrategia de predicción de saltos basada en la historia de los saltos se rechazó porque los diseñadores pensaron que se conseguiría un beneficio mínimo.

Como ejemplo del efecto de la predicción de saltos, considere el programa de la Figura 14.13 y suponga que el procesador de saltos predice que el salto condicional no se producirá (el caso implícito para un salto hacia delante). La Figura 14.14a muestra el efecto en el cauce si de hecho el salto no se produce. En el primer ciclo, la cola de envío se carga con ocho instrucciones. Las seis primeras instrucciones son instrucciones de enteros y se envían, una en cada ciclo, a la unidad de enteros. La instrucción de salto condicional no se puede enviar hasta que llegue a la mitad inferior de la cola de envío, lo que sucede en el ciclo 5. La unidad de saltos predice que este salto no se producirá, y por tanto la siguiente instrucción secuencial se envía condicionalmente (indicada como D’). El salto no puede resolverse hasta que se ejecute la instrucción de comparación en el ciclo 8. En ese momento, el procesador de saltos confirma que la predicción fue correcta, y la ejecución continúa. No hay retardos y el cauce se mantiene lleno.

Observe que no se captan instrucciones en los ciclos del 4 al 8. Esto es debido a que la caché está ocupada durante esos ciclos con la etapa de acceso a la caché de las cinco instrucciones de carga. Incluso así, el flujo de instrucciones no se retrasa, porque la cola de envío puede contener ocho instrucciones.

La Figura 14.14b muestra el resultado si la predicción es incorrecta y el salto se produce. En este caso, las tres instrucciones que comienzan en el IF se tienen que desechar, y la captación se reanuda en las instrucciones que comienzan en el ELSE. Por consiguiente, la etapa de ejecución del cauce de enteros está desocupada en los ciclos 9 y 10, perdiéndose dos ciclos por culpa de la predicción incorrecta.

POWERPC 620

El 620 es la primera implementación de 64 bits de la arquitectura PowerPC. Una característica notable de esta implementación es que incluye seis unidades de ejecución independientes:

```
        if (a > 0)
                a = a + b + c + d + e;
        else
                a = a – b – c – d – e;
```

(a) Código C

```
                                     #r1 apunta a a,
                                     #r1+4 apunta a b,
                                     #r1+8 apunta a c,
                                     #r1+12 apunta a d,
                                     #r1+16 apunta a e.
       lwz    r8=a(r1)               #cargar a
       lwz    r12=b(r1,4)            #cargar b
       lwz    r9=c(r1,8)             #cargar c
       lwz    r10=d(r1,12)           #cargar d
       lwz    r11=e(r1,16)           #cargar e
       cmpi   cr0=r8,0               #comparación inmediata
       bc     ELSE,cr0/gt=false      #saltar si el bit indica falso
IF:
       add    r12=r8,r12             #sumar
       add    r12=r12,r9             #sumar
       add    r12=r12,r10            #sumar
       add    r4=r12,r11             #sumar
       stw    a(r1)=r4               #almacenar
       b      OUT                    #salto incondicional
ELSE:
       subf   r12=r12,r8             #restar
       subf   r12=r9,r12             #restar
       subf   r12=r10,r12            #restar
       subf   r4=r12,r11             #restar
       stw    a(r1)=r4               #almacenar
OUT:
```

(b) Código ensamblador

Figura 14.13. Ejemplo de código con un salto condicional [WEIS94].

- Unidad de instrucciones.
- Tres unidades de enteros.
- Unidad de carga/almacenamiento.
- Unidad de coma flotante.

Esta organización permite al procesador enviar hasta cuatro instrucciones simultáneamente a las tres unidades de enteros y a la de coma flotante.

El 620 emplea una estrategia de predicción de saltos de altas prestaciones que incluye lógica de predicción, buffers de renombramiento de registros, y centrales de reserva dentro de las unidades de ejecución. Cuando se capta una instrucción, se le asigna un buffer de renombramiento que contiene temporalmente los resultados de la instrucción, tales como los datos a almacenar en registros. Gracias al uso de los buffers de renombramiento, el procesador puede *ejecutar especulativamente* instrucciones basadas en predicción de saltos; si la predicción resulta ser incorrecta, los resultados de las instrucciones especulativas pueden desecharse sin afectar al banco de registros. Una vez confirmado el resultado de un salto, los resultados temporales pueden escribirse de modo definitivo.

```
                            1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16
     lwz  r8=a(r1)          F    D    E    C    W
     lwz  r12=b(r1,4)       F    .    D    E    C    W
     lwz  r9=c(r1,8)        F    .    .    D    E    C    W
     lwz  r10=d(r1,12)      F    .    .    .    D    E    C    W
     lwz  r11=e(r1,16)      F    .    .    .    .    D    E    C    W
     cmpi cr0=r8,0          F    .    .    .    .    .    D    E
     bc   ELSE,cr0/gt=false F    .    .    .    S
IF:  add  r12=r8,r12        F    .    .    .    .    .    .    D'   E    W
     add  r12=r12,r9                  F    .    .    .    .    .    D    E    W
     add  r12=r12,r10                 F    .    .    .    .    .    .    D    E    W
     add  r4=r12,r11                                                 F    .    D    E    W
     stw  a(r1)=r4                                                   F    .    .    D    E    C
     b    OUT
ELSE:subf r12=r8,r12
     subf r12=r12,r9
     subf r12=r12,r10
     subf r4=r12,r11
     stw  a(r1)=r4
OUT:
```

(a) Predicción correcta: el salto no se produjo

```
                            1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16
     lwz  r8=a(r1)          F    D    E    C    W
     lwz  r12=b(r1,4)       F    .    D    E    C    W
     lwz  r9=c(r1,8)        F    .    .    D    E    C    W
     lwz  r10=d(r1,12)      F    .    .    .    D    E    C    W
     lwz  r11=e(r1,16)      F    .    .    .    .    D    E    C    W
     cmpi cr0=r8,0          F    .    .    .    .    .    D    E
     bc   ELSE,cr0/gt=false F    .    .    .    S
IF:  add  r12=r8,r12        F    .    .    .    .    .    .    D'
     add  r12=r12,r9                  F
     add  r12=r12,r10                 F    .    .    .    .    .
     add  r4=r12,r11
     stw  a(r1)=r4
     b    OUT
ELSE:subf r12=r8,r12                                                 F    D    E    W
     subf r12=r12,r9                                                 F    .    D    E    W
     subf r12=r12,r10                                                F    .    .    D    E    W
     subf r4=r12,r11                                                 F    .    .    .    D    E    W
     stw  a(r1)=r4                                                   F    .    .    .    .    D    E    C
OUT:
```

(b) Predicción incorrecta: el salto de produjo

F = captación
D = envío/decodificación
E = ejcución/direccionamiento
C = acceso a caché
W = escritura
S = envío

Figura 14.14. Predicción de salto: no se producirá el salto [WEIS94].

Cada unidad tiene dos o más centrales de reserva, que almacenan aquellas instrucciones enviadas que deben suspenderse en espera de resultados de otras instrucciones. Esta característica posibilita que se retiren estas instrucciones de la unidad de instrucciones, permitiendo a esta continuar enviando instrucciones a las otras unidades de ejecución.

El 620 puede ejecutar especulativamente hasta cuatro instrucciones de salto no resueltas (frente a una en el 601). La predicción de saltos se basa en el uso de una tabla de historia de saltos con 2 048 elementos. Las simulaciones ejecutadas por los diseñadores del PowerPC muestran que la tasa de aciertos en la predicción de saltos es del noventa por ciento [THOM94].

14.5. LECTURAS RECOMENDADAS

Dos buenas obras sobre el diseño superescalar son [SHEN05] y [OMON99]. [SMIT95] y [SIMA97] son artículos generales sobre la materia dignos de consideración. [JOUP89a] examina el paralelismo en las instrucciones, explora varias técnicas para maximizar el paralelismo, y compara las aproximaciones superescalar y supersegmentada usando simulaciones. Entre los artículos recientes que proporcionan una buena cobertura sobre cuestiones de diseño superescalar se encuentran [SIMA04], [PATT01] y [MOSH01].

[POPE91] proporciona un estudio detallado de una máquina superescalar propuesta. También ofrece una excelente revisión de los temas de diseño relacionados con políticas de ejecución desordenada de instrucciones. En [KUGA91] se encuentra otro estudio de un sistema propuesto; este artículo plantea y estudia la mayoría de las cuestiones importantes sobre el diseño de las implementaciones superescalares. [LEE91] examina las técnicas software que se pueden usar para aumentar las prestaciones superescalares. [WALL91] es un interesante estudio del grado hasta el cual puede explotarse el paralelismo en las instrucciones en un procesador superescalar.

El Volumen I de [INTE04] ofrece una descripción general del cauce del Pentium 4; más detalles se pueden consultar en [INTE01a] e [INTE01b].

[POTT94] incluye una revisión detallada de la segmentación de instrucciones del PowerPC 601. [SHAN95] también ofrece una buena cobertura.

HINT01 HINTON, G. *et al.*: «The Microarchitecture of the Pentium 4 Processor». *Intel Technology Journal,* Q1 2001. **http://developer.intel.com/technology/itj/**

INTE04 Intel Corp. IA-32 *Intel Architecture Software Developer's Manual (4 volúmenes).* Documento 253665 hasta 253668. 2004. **http://developer.intel.com/design/Pentium4/documentation.htm**

INTE01a Intel Corp. *Intel Pentium 4 Processor Optimization Reference Manual.* Documento 248966-04 2001. **http://developer.intel.com/design/Pentium4/documentation.htm**

INTE01b Intel Corp. *Desktop Performance and Optimization for Intel Pentium 4 Processor.* Documento 248966-04 2001. **http://developer.intel.com/design/Pentium4/documentation.htm**

JOUP89a JOUPPI, N. y WALL, D.: «Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines». *Proceedings, Third International Conference on Architectural Support for Programming Languages and Operating Systems,* abril, 1989.

KUGA91 KUGA, M.; MURAKAMI, K. y TOMITA, S.: «DSNS (Dynamically-hazard resolved, Statically-code-scheduled, Nonuniform Superscalar): Yet Another Superscalar Processor Architecture». *Computer Architecture News,* junio, 1991.

LEE91 LEE, R.; KWOK, A. y BRIGGS, F.: «The Floating Point Performance of a Superscalar SPARC Processor». *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems,* abril, 1991.

MOSH01 MOSHOVOS, A. y SOHI, G.: «Microarchitectural Innovations: Boosting Microprocessor Performance Beyond Semiconductor Technology Scaling». *Proceedings of the IEEE,* noviembre, 2001.

OMON99 OMONDI, A.: *The Microarchitecture of Pipelined and Superscalar Computers.* Boston. Kluwer, 1999.

PATT01 PATT, Y.: «Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution». Reduced Instruction Set Computer Architecture." *Proceedings of the IEEE,* noviembre, 2001.

POPE91 POPESCU, V. y *et al.*: «The Metaflow Architecture». *IEEE Micro,* junio, 1991.

POTT94 POTTER, T. *el al.*: «Resolution of Data and Control-Flow Dependencies in the PowerPC 601». *IEEE Micro,* octubre, 1994.

SHAN95 SHANLEY, T.: *PowerPC System Architecture.* Reading, MA. Addison-Wesley, 1995.

SHEN05 SHEN, J. y LIPASTI, M.: *Modern Processor Design: Fundamentals of Superscalar Processors.* Nueva York. McGraw-Hill, 2005.

SIMA97 SIMA, D.: «Superscalar Instruction Issue». *IEEE Micro,* septiembre/octubre, 1997.

SIMA04 SIMA, D.: «Decisive Aspects in the Evolution of Microprocessors». *Proceedings of the IEEE,* diciembre, 2004.

SMIT95 SMITH, J. y SOHI, G.: «The Microarchitecture of Superscalar Processors». *Proceedings of the IEEE,* diciembre, 1995.

WALL91 WALL, D.: «Limits of Instruction-Level Parallelism». *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems,* abril, 1991.

14.6. PALABRAS CLAVE, PREGUNTAS DE REPASO Y PROBLEMAS

PALABRAS CLAVE

antidependencia
conflicto en los recursos
dependencia de datos verdadera
dependencia de flujo
dependencia de salida
dependencia escritura-escritura dependencia escritura-lectura
dependencia lectura-escritura
dependencia relativa al procedimientoemisión de instrucciones
emisión desordenada
emisión en orden
entrega
finalización desordenada
finalización en orden
micro-ops
microoperaciones
paralelismo de la máquina
paralelismo en las instrucciones predicción de saltos
renombramiento de registros
retiro
superescalar
supersegmentado
ventana de instrucciones

PREGUNTAS DE REPASO

14.1. ¿Cuál es la característica esencial del enfoque superescalar en el diseño del procesador?

14.2. ¿Cuál es la diferencia entre las aproximaciones superescalar y supersegmentada?

14.3. ¿Qué es el paralelismo en las instrucciones?

14.4. Defina brevemente los siguientes términos:

- Dependencia de datos verdadera
- Dependencia relativa al procedimiento
- Conflictos en los recursos
- Dependencia de salida
- Antidependencia

14.5. ¿Cuál es la distinción entre el paralelismo en las instrucciones y el paralelismo de la máquina?

14.6. Enumere y defina brevemente tres tipos de políticas de emisión superescalar de instrucciones.

14.7. ¿Cuál es la finalidad de una ventana de instrucciones?

14.8. ¿Qué es el renombramiento de registros y cuál es su propósito?

14.9. ¿Cuáles son los elementos clave de la organización superescalar del procesador?

PROBLEMAS

14.1. Cuando se usa finalización desordenada en un procesador superescalar, la reanudación de la ejecución después del procesamiento de una interrupción es complicada, porque la condición de excepción puede haberse detectado en una instrucción que produjo su resultado fuera de orden. El programa no puede reanudarse en la instrucción siguiente a la instrucción de la excepción, porque las siguientes instrucciones ya han finalizado, y hacerlo así provocaría que esas instrucciones se ejecutaran dos veces. Sugiera un mecanismo o varios para tratar esta situación.

14.2. Considere la siguiente secuencia de instrucciones, donde la sintaxis consta de un código de operación seguido de un registro destino, seguido a su vez por uno o dos registros fuente:

```
0    ADD    R3, R1, R2
1    LOAD   R6, [R3]
2    AND    R7, R5, 3
3    ADD    R1, R0, R0
4    SRL    R7, R0, 8
5    OR     R2, R4, R7
6    SUB    R5, R3, R4
7    ADD    R0, R1, R10
8    LOAD   R6, [R5]
9    SUB    R2, R1, R6
10   AND    R3, R7, 15
```

Suponga el uso de un cauce de cuatro etapas: captación, decodificación/emisión, ejecución, y escritura. Suponga que todas las etapas del cauce tardan un ciclo de reloj excepto la etapa de ejecución. En las instrucciones aritméticas y lógicas con enteros sencillas, la etapa de ejecución necesita un ciclo, pero un LOAD desde memoria consume cinco ciclos en dicha etapa.

Si tenemos un cauce escalar sencillo pero que permite ejecución desordenada, podemos construir la siguiente tabla para la ejecución de las siete primeras instrucciones:

Instrucción	Captación	Decodificación	Ejecución	Escritura
0	0	1	2	3
1	1	2	4	9
2	2	3	5	6
3	3	4	10	11
4	4	5	6	7
5	5	6	8	10
6	6	7	9	12

Los elementos bajo las cuatro etapas del cauce indican el ciclo de reloj en el que cada instrucción inicia cada fase. En este programa, la segunda instrucción ADD (instrucción 3) depende de la instrucción LOAD (instrucción 1) en uno de sus operados, r6. Como la instrucción LOAD tarda cinco ciclos de reloj, y la lógica de emisión encuentra la instrucción dependiente ADD después de dos ciclos, la lógica de emisión tiene que retrasar la instrucción ADD tres ciclos de reloj. Con una capacidad de ejecución desordenada, el procesador puede detener la instrucción 3 en el ciclo de reloj 4, y pasar a emitir las siguientes tres instrucciones independientes, que entran en ejecución en los ciclos 6, 8 y 9.

La instrucción LOAD termina su ejecución en el ciclo 9, y entonces la instrucción dependiente ADD puede comenzar su ejecución en el ciclo 10.

(a) Complete la tabla anterior.
(b) Rehaga la tabla suponiendo que no se tiene la capacidad de ejecución desordenada. ¿Cuál es el ahorro usando esa capacidad?
(c) Rehaga la tabla suponiendo una implementación superescalar que pueda manejar dos instrucciones a la vez en cada etapa.

14.3. En la cola de instrucciones de la unidad de envío del PowerPC 601, las instrucciones pueden enviarse desordenadamente a las unidades de procesamiento de saltos y de coma flotante, pero las instrucciones dirigidas a la unidad de enteros tienen que ser enviadas solo desde el elemento inferior de la cola. ¿Por qué existe esta limitación?

14.4. Elabore una figura similar a la Figura 14.14 para los siguientes casos:

(a) Predicción de salto: se producirá el salto; predicción correcta: se produce el salto.
(b) Predicción de salto: se producirá el salto; predicción incorrecta: el salto no se produce.

14.5. Considere el siguiente programa en lenguaje ensamblador:

```
I1: Move R3, R7          /R3 ← (R7)/
I2: Load R8, (R3)        /R8 ← Memoria (R3)/
I3: Add R3, R3, 4        /R3 ← (R3) + 4/
I4: Load R9, (R3)        /R9 ← Memoria (R3)/
I5: BLE R8, R9, L3       /Saltar si (R9) > (R8)/
```

Este programa incluye dependencias escritura-escritura, lectura-escritura, y escritura-lectura. Muéstrelas.

14.6. La Figura 14.15 muestra un ejemplo de organización superescalar de un procesador. El procesador puede emitir dos instrucciones por ciclo si no hay conflicto por los recursos ni problemas de dependencias de datos. Hay básicamente dos cauces, con cuatro etapas de procesamiento (captación, decodificación, ejecución y almacenamiento). Cada cauce tiene su propia unidad de captación, decodificación y almacenamiento. Hay disponibles cuatro unidades funcionales (multiplicador, sumador, unidad lógica y unidad de carga) para la etapa de ejecución, que son compartidas por los dos cauces de forma dinámica. Los dos cauces pueden usar dinámicamente las dos unidades de almacenamiento, dependiendo de su disponibilidad en un ciclo concreto. Hay una ventana de anticipación con su propia lógica de captación y decodificación. Esta ventana se usa para la búsqueda anticipada de instrucciones de cara a una emisión desordenada.

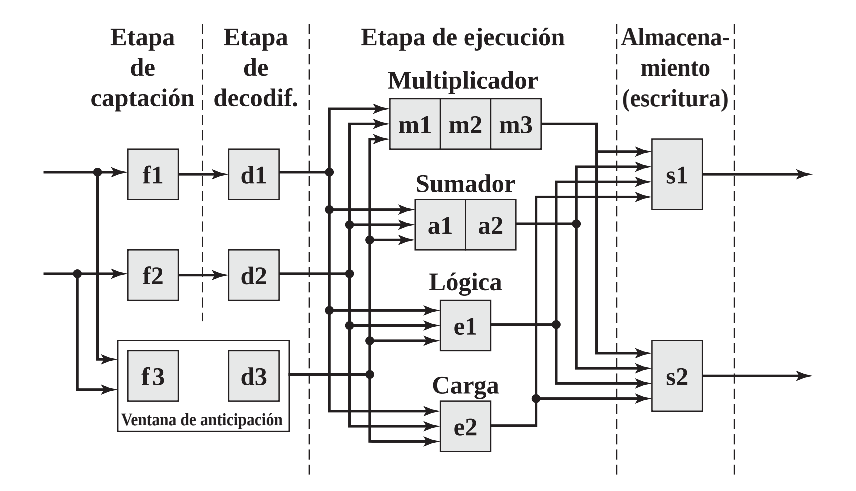


Figura 14.15. Un procesador superescalar con un cauce doble.

Considere el siguiente programa que va a ejecutarse en este procesador:

```
I1: Load R1, A        /R1 ← Memoria (A)/
I2: Add R2, R1        /R2 ← (R2) + (R1)/
I3: Add R3, R4        /R3 ← (R3) + (R4)/
I4: Mul R4, R5        /R4 ← (R4) + (R5)/
I5: Comp R6           /R6 ← (R6)/
I6: Mul R6, R7        /R6 ← (R6) × (R7)/
```

(a) ¿Qué dependencias hay en el programa?

(b) Muestre la actividad del cauce para este programa en el procesador de la Figura 14.15 con políticas de emisión en orden y finalización en orden, usando una presentación similar a la de la Figura 14.2.

(c) Repita el apartado anterior para emisión en orden y finalización desordenada.

(d) Repita el apartado anterior para emisión desordenada y finalización desordenada.

14.7. La Figura 14.16 está tomada de un artículo sobre diseño superescalar. Explique las tres partes de la figura, y defina w, x, y, y z.

14.8. El algoritmo de predicción dinámica de saltos de Yeh, usado en el Pentium 4, es un algoritmo de predicción de saltos en dos niveles. El primer nivel es la historia de los últimos *n* saltos. El segundo nivel es el comportamiento de los saltos de las últimas *s* ocurrencias del patrón único de los últimos *n* saltos. Está implementado como ha sido posible. Para cada instrucción de salto condicional del programa, hay un elemento en la tabla de historia de saltos (*Branch History Table*, BHT). Cada elemento consta de *n* bits correspondientes a las últimas *n* ejecuciones de la instrucción de salto, con un 1 si el salto se produjo y un 0 si no se produjo. Cada elemento de la BHT indexa una tabla de patrones (*Pattern Table*, PT) que tiene 2^n elementos, uno por cada posible patrón de *n* bits. Cada elemento de la PT consta de s bits, usados en la predicción de saltos, como se describió en el Capítulo 12 (por ejemplo, en la Figura 12.17). Cuando se encuentra un salto condicional durante la captación y decodificación de una instrucción, la dirección de la instrucción se usa para acceder al elemento adecuado de la BHT, que muestra la historia reciente de la instrucción. Entonces, el elemento de la BHT se usa para acceder al elemento adecuado de la PT para realizar la predicción de salto. Después de que se ejecute el salto, se actualiza el elemento de la BHT, y después se actualiza el elemento adecuado de la PT.

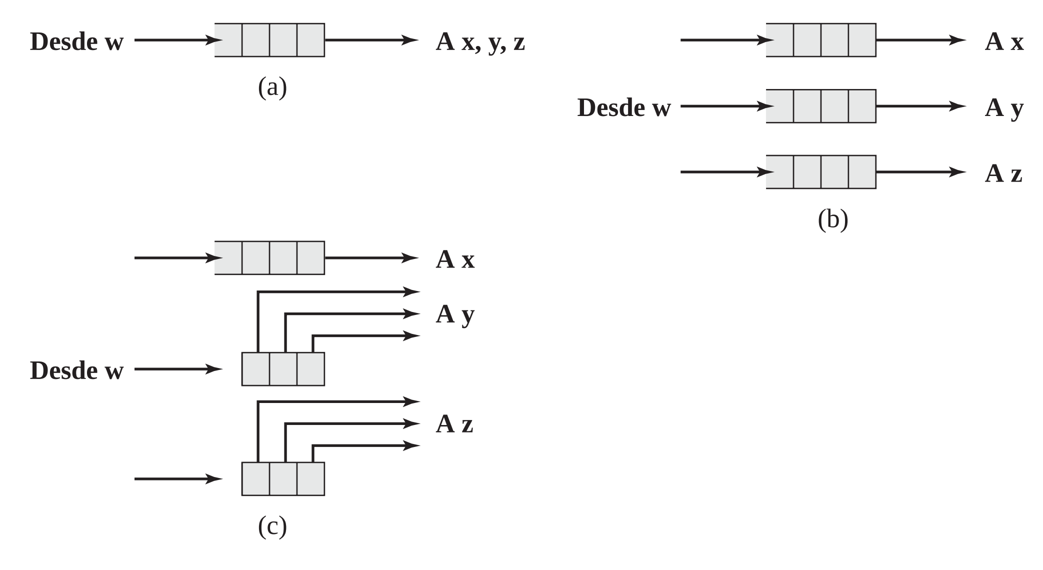


Figura 14.16. Figura del Problema 14.7.

(a) Durante el test de prestaciones de este esquema, Yeh probó cinco esquemas de predicción, ilustrados en la Figura 14.17. Identifique qué tres esquemas de los cinco corresponden a los que se muestran en las Figuras 12.17 y 12.25. Describa los dos esquemas restantes.

(b) Con este algoritmo, la predicción no se basa solo en la historia reciente de una instrucción de salto particular. En lugar de ello, se basa en la historia de todos los patrones de salto que corresponden al patrón de *n* bits en el elemento de la BHT para esa instrucción. Sugiera una razón fundamental para usar esta estrategia.

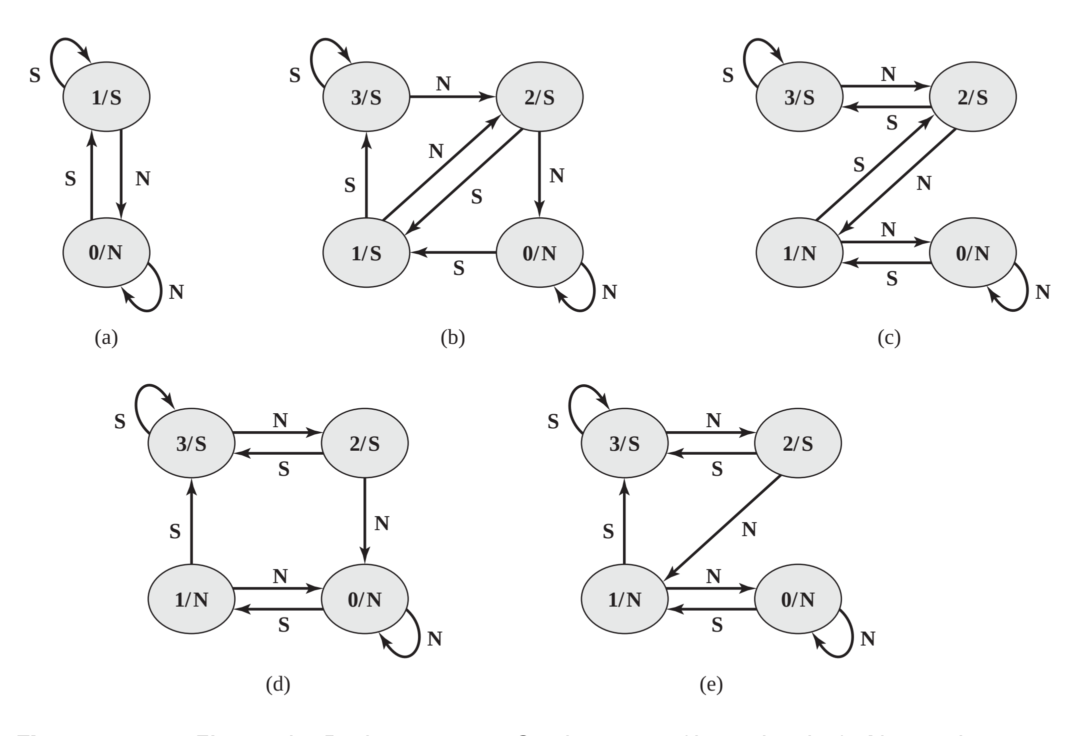


Figura 14.17. Figura del Problema 14.8. *S* = hay salto (*branch taker*); *N* = no hay salto (*branch not taken*).

CAPÍTULO 15

La arquitectura IA-64

PUNTOS CLAVE

- La arquitectura de conjunto de instrucciones IA-64 es una nueva solución para proporcionar soporte desde el hardware al paralelismo en las instrucciones y es significativamente distinta a la aproximación seguida en las arquitecturas superescalares.
- Las características más destacables de la arquitectura IA-64 son el soporte hardware a la ejecución con predicados, la especulación en el control, la especulación en los datos y la segmentación software.
- Con la **ejecución con predicados**, cada instrucción de IA-64 incluye una referencia a un registro de predicado de un bit, y solo se ejecuta si el valor del predicado es 1 (verdadero). Ello permite al procesador ejecutar especulativamente las dos ramas de una sentencia **if** y entregar el resultado solo cuando la condición queda determinada.
- Con la **especulación en el control**, una instrucción de carga se coloca en un lugar previo del programa y la posición original se reemplaza por una instrucción de comprobación. La carga prematura ahorra tiempo de ciclo; si la carga produce una excepción, esta no se activa hasta que la instrucción de comprobación determina si la carga debería haberse producido.
- Con la **especulación en los datos**, una carga se coloca antes de una instrucción de almacenamiento que podría alterar la posición de memoria fuente de la carga. Una comprobación posterior se encarga de asegurar que la carga reciba el valor de memoria correcto.
- La **segmentación software** es una técnica en la cual las instrucciones de distintas iteraciones de un bucle pueden ejecutarse en paralelo.

Con el Pentium 4, la familia de microprocesadores que comenzó con el 8086, y que ha sido la línea de productos informáticos de mayor éxito de todos los tiempos, parece haber llegado a su fin. Intel se ha asociado con Hewlett-Packard (HP) para desarrollar una nueva arquitectura de 64 bits, llamada IA-64. La arquitectura IA-64 no es una ampliación a 64 bits de la arquitectura de 32 bits x86 de Intel, ni una adaptación de la arquitectura PA-RISC de 64 bits de Hewlett-Packard. En lugar de ello, la IA-64 es una nueva arquitectura edificada sobre años de investigación en las dos compañías y en algunas universidades. La arquitectura aprovecha la enorme circuitería y la gran velocidad disponibles en las más recientes generaciones de microchips gracias a la utilización sistemática del paralelismo. La arquitectura IA-64 representa una importante novedad respecto a la tendencia hacia esquemas superescalares que ha dominado el desarrollo reciente de los procesadores.

Comenzamos este capítulo con una discusión sobre los factores que han motivado la nueva arquitectura. A continuación, estudiamos la organización general necesaria para dar soporte a la arquitectura. Después examinamos con cierto detalle las características clave que posee la arquitectura IA-64 para facilitar el paralelismo en las instrucciones. Por último, estudiamos la arquitectura de conjunto de instrucciones IA-64 y la organización del Itanium.

15.1. MOTIVACIÓN

Los conceptos básicos en los que se fundamenta la arquitectura IA-64 son los siguientes:

- Paralelismo en las instrucciones que queda explícito en las instrucciones máquina, en lugar de depender del procesador en tiempo de ejecución.
- Palabras de instrucción largas o muy largas (*long instruction word,* LIW / *very long instruction word,* VLIW).
- Ejecución de saltos basada en predicados (concepto diferente al de predicción de saltos).
- Carga especulativa.

Intel y HP hacen referencia a esta combinación de conceptos con el nombre de computación con instrucciones explícitamente paralelas (*Explicitly Parallel Instruction Computing,* EPIC). Intel y HP utilizan el término **EPIC** para referirse a la tecnología o al conjunto de técnicas. La arquitectura **IA-64** consiste en un repertorio de instrucciones real destinado a ser implementado usando la tecnología EPIC. El primer producto de Intel basado en IA-64 es conocido con el nombre de **Itanium**. Le seguirán otros productos basados en la misma arquitectura IA-64.

La Tabla 15.1 resume las principales diferencias entre la arquitectura IA-64 y la aproximación superescalar tradicional.

Para Intel, mudarse a una nueva arquitectura, con un hardware incompatible con la arquitectura de instrucciones x86, supuso una decisión muy crítica. Pero estaba impulsada por los dictados de la tecnología. Cuando comenzó la familia x86, hacia finales de los setenta, el procesador tenía de decenas a miles de transistores y era un dispositivo esencialmente escalar. Es decir, se procesaba una instrucción en cada instante, con poca o ninguna segmentación. Cuando el número de transistores creció a cientos de miles a mediados de los ochenta, Intel introdujo la segmentación (por ejemplo, *ver* Figura 12.19). Mientras tanto, otros fabricantes intentaban sacar partido al creciente número de transistores y aumentaban la velocidad por medio del enfoque RISC, que permitía una segmentación más

Tabla 15.1. Arquitectura superescalar tradicional frente a IA-64.

Superescalar	IA-64
Instrucciones de tipo RISC, una por palabra	Instrucciones de tipo RISC puestas en grupos de tres
Múltiples unidades de ejecución en paralelo	Múltiples unidades de ejecución en paralelo
Reordena y optimiza el flujo de instrucciones en tiempo de ejecución	Reordena y optimiza el flujo de instrucciones en tiempo de compilación
Predicción de saltos con ejecución especulativa de un camino	Ejecución especulativa de los dos caminos de una bifurcación
Carga datos desde memoria solo cuando es necesario, e intenta encontrar los datos primero en las cachés	Carga datos especulativamente antes de que se necesiten, y sigue intentando encontrar los datos primero en las cachés

eficiente, y más tarde por medio de la combinación superescalar/RISC, que traía consigo múltiples unidades de ejecución. Con el Pentium, Intel hizo un intento moderado de utilizar técnicas superescalares, permitiendo que dos instrucciones CISC se ejecutaran al mismo tiempo. Después, el Pentium Pro y el Pentium II incorporaron una traducción de instrucciones CISC a microoperaciones de tipo RISC y un uso más agresivo de técnicas superescalares. Este enfoque permitió la utilización eficaz de un chip con millones de transistores. Pero para el procesador de la siguiente generación, el que viene después de los Pentium, Intel y otros fabricantes se enfrentan a la necesidad de utilizar eficazmente decenas de millones de transistores en un único chip procesador.

Los diseñadores de procesadores tienen pocas alternativas a la hora de usar esta abundancia de transistores. Una posibilidad es dedicar esos transistores adicionales a mayores cachés internas. Esto puede mejorar las prestaciones hasta cierto punto, pero llega un momento en el que aumentar la caché se traduce en una mejora insignificante de la tasa de aciertos. Otra alternativa es colocar varios procesadores en el mismo chip. Esta solución se discute en los Capítulos 2 y 16. Otra alternativa más es incrementar el grado de paralelismo superescalar añadiendo más unidades de ejecución. El problema de esta solución es que los diseñadores están tropezando, de hecho, contra un muro de complejidad. Conforme se añaden más y más unidades de ejecución, haciendo el procesador más «ancho», se necesita más lógica para organizarlas. Ha de mejorarse la predicción de saltos, hay que usar procesamiento sin orden y hay que emplear cauces más largos. Pero con cauces cada vez más largos, hay una mayor penalización por fallos en la predicción. La ejecución desordenada requiere un gran número de registros de renombramiento y una compleja circuitería para acabar con las dependencias. La consecuencia es que los mejores procesadores de hoy día pueden retirar como mucho seis instrucciones por ciclo, y por lo general menos.

Para solucionar estos problemas, Intel y HP han propuesto un planteamiento de diseño global que permite la utilización eficaz de un procesador con muchas unidades de ejecución en paralelo. El corazón de este nuevo enfoque es el concepto de paralelismo explícito. En esta aproximación, el compilador planifica estáticamente las instrucciones en tiempo de compilación, en lugar de que lo haga dinámicamente el procesador en tiempo de ejecución. El compilador determina qué instrucciones pueden ejecutarse en paralelo e incluye esta información en la instrucción máquina. El procesador usa esa información para llevar a cabo la ejecución paralela. Una ventaja de esta aproximación es que el procesador EPIC no requiere tanta circuitería compleja como un procesador superescalar capaz de ejecutar instrucciones sin orden. Además, mientras que el procesador tiene que determinar la posibilidad de una potencial ejecución en paralelo en cuestión de nanosegundos, el compilador dispone de un plazo varios órdenes de magnitud mayor para examinar el código con detenimiento y estudiar el programa globalmente.

15.2. ORGANIZACIÓN GENERAL

Como cualquier arquitectura de procesador, la IA-64 puede implementarse con diversas organizaciones. La Figura 15.1 indica en términos generales la organización de una máquina IA-64. Sus características más importantes son las siguientes:

- **Un gran número de registros:** el formato de instrucción de la arquitectura IA-64 supone el empleo de 256 registros, 128 registros de 64 bits de uso general para uso con enteros, con datos lógicos y para propósito general, y 128 registros de 82 bits para uso con coma flotante

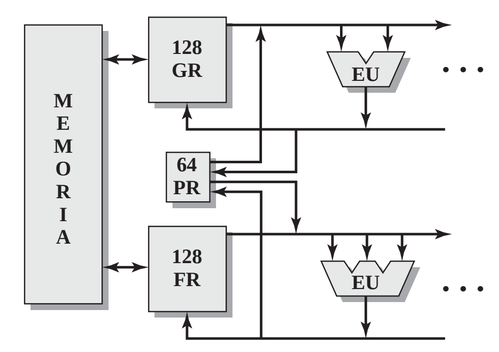


Figura 15.1. Organización general de la arquitectura IA-64.

y gráficos. También hay 64 registros de predicado de un bit, usados para la ejecución con predicados como se explica más tarde.

- **Múltiples unidades de ejecución:** una máquina superescalar comercial típica puede tener hoy día cuatro cauces paralelos, empleando cuatro unidades de ejecución en paralelo tanto para la parte de enteros del procesador como para la de coma flotante. Se espera que la IA-64 se implemente en sistemas con ocho o más unidades paralelas.

El banco de registros es bastante grande comparado con la mayoría de las máquinas RISC y superescalares. La razón de ello es que se necesita un número de registros grande para permitir un alto grado de paralelismo. En una máquina superescalar tradicional, el lenguaje máquina (y el lenguaje ensamblador) emplea un pequeño número de registros visibles, y el procesador establece una correspondencia con un número mayor de registros usando técnicas de renombramiento y análisis de dependencias. Dado que deseamos tener paralelismo explícito y liberar al procesador de la carga que supone el renombramiento de registros y el análisis de dependencias, necesitamos un gran número de registros explícitos.

El número de unidades de ejecución depende del número de transistores disponibles en una implementación concreta. El procesador explotará el paralelismo hasta el punto que pueda. Por ejemplo, si el flujo de instrucciones en lenguaje máquina indica que se pueden ejecutar ocho instrucciones de enteros en paralelo, un procesador con cuatro cauces de enteros las ejecutará en dos turnos. Un procesador con ocho cauces ejecutará las ocho instrucciones simultáneamente.

En la arquitectura IA-64 se definen cuatro tipos de unidades de ejecución:

- **Unidad I:** para instrucciones aritméticas con enteros, de desplazamiento y suma, lógicas, de comparación, y multimedia con enteros.
- **Unidad M:** cargas y almacenamientos entre registros y memoria más algunas operaciones de la ALU con enteros.
- **Unidad B:** instrucciones de salto.
- **Unidad F:** instrucciones de coma flotante.

Tabla 15.2. Relación entre tipo de instrucción y unidad de ejecución.

Tipo de instrucción	Descripción	Tipo de unidad de ejecución
A	Enteros ALU	Unidad I o unidad M
I	Enteros no-ALU	Unidad I
M	Memoria	Unidad M
F	Coma flotante	Unidad F
B	Salto	Unidad B
X	Ampliada	Unidad I / unidad B

Las instrucciones IA-64 se clasifican en seis tipos. La Tabla 15.2 enumera los tipos de instrucciones y los tipos de unidades de ejecución en las que pueden ejecutarse. El tipo de instrucción ampliada (X) incluye instrucciones en las cuales se usan dos posiciones de un paquete para codificar la instrucción, permitiendo más información de la que cabe en una instrucción de 41 bits (los paquetes se explican en la próxima sección).

15.3. USO DE PREDICADOS, ESPECULACIÓN, Y SEGMENTACIÓN DE SOFTWARE

Esta sección estudia las características principales de la arquitectura IA-64 que sirven de soporte al paralelismo en las instrucciones. En primer lugar, es necesario ofrecer una visión de conjunto del formato de instrucción IA-64 y, para entender los ejemplos de esta sección, definir el formato general de las instrucciones en lenguaje ensamblador de IA-64.

FORMATO DE INSTRUCCIÓN

La arquitectura IA-64 define un **paquete** de 128 bits que contiene tres instrucciones, llamadas **sílabas**, y un campo plantilla (Figura 15.2a). El procesador puede captar uno o más paquetes de instrucciones a la vez; y cada captación de paquete contiene tres instrucciones. El campo plantilla contiene información que indica qué instrucciones se pueden ejecutar en paralelo. La interpretación de este campo no se limita a un único paquete. Por el contrario, el procesador puede examinar varios paquetes para determinar qué instrucciones pueden ejecutarse en paralelo. Por ejemplo, el flujo de instrucciones puede ser de tal tipo que permita la ejecución en paralelo de ocho instrucciones. El compilador reordenará las instrucciones de manera que las ocho abarquen paquetes contiguos, y ajustará los bits de la plantilla de forma que el procesador sepa que las ocho instrucciones son independientes.

Las instrucciones agrupadas no tienen que estar en el orden original del programa. Además, debido a la flexibilidad del campo plantilla, el compilador puede mezclar instrucciones dependientes e independientes en el mismo paquete. Al contrario que algunos diseños VLIW previos, la

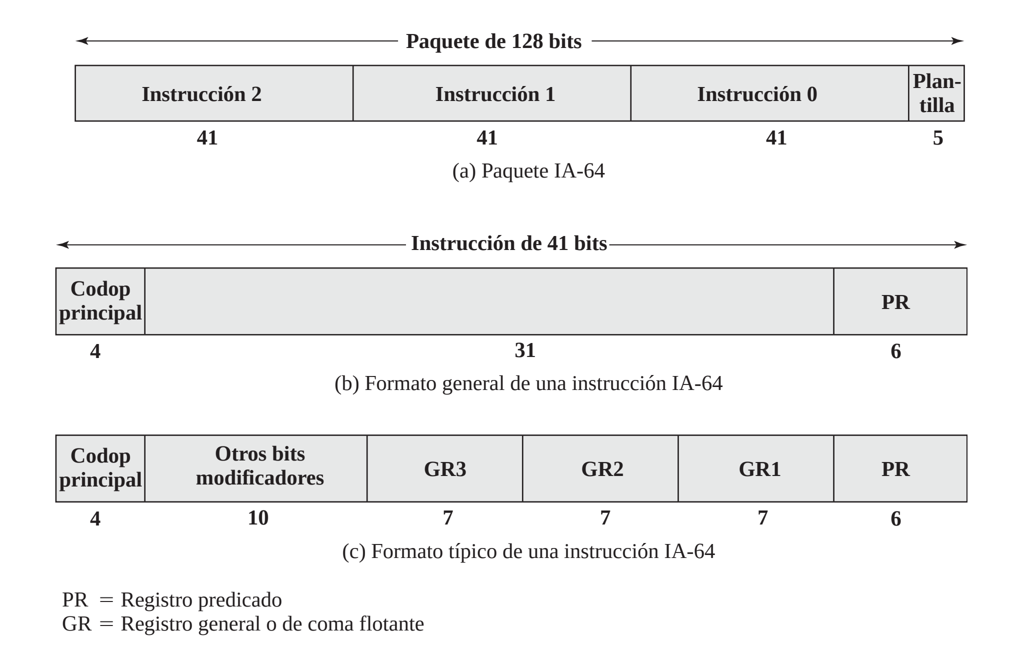


Figura 15.2. Formato de instrucción de la arquitectura IA-64.

arquitectura IA-64 no necesita insertar instrucciones de no operación (NOP) para completar los paquetes.

La Tabla 15.3 muestra la interpretación de los posibles valores del campo plantilla, de cinco bits (algunos valores están reservados y no se usan actualmente). El valor de la plantilla tiene dos propósitos:

1. El campo especifica la correspondencia de las instrucciones con los tipos de unidades de ejecución. No son posibles todas las correspondencias de instrucciones con unidades.
2. El campo indica la presencia de posibles **paradas**. Una parada indica al hardware que una o más instrucciones anteriores a la parada pueden tener ciertos tipos de dependencias de recursos con una o más instrucciones posteriores a la parada. En la tabla, una línea vertical gruesa indica una parada.

Cada instrucción tiene un formato de longitud fija de 41 bits (Figura 15.2b). Esta longitud es algo mayor que la tradicional de 32 bits de las máquinas RISC y RISC superescalares (aunque es mucho más corta que la microoperación de 118 bits del Pentium 4). Hay dos factores que conducen a estos bits añadidos. En primer lugar, la arquitectura IA-64 utiliza más registros que una máquina RISC típica: 128 para enteros y 128 para coma flotante. En segundo lugar, para dar cabida a la técnica de ejecución con predicados, una máquina IA-64 incluye 64 registros de predicado. Su uso se explica más adelante.

La Figura 15.2c muestra con mayor detalle el formato de instrucción típico. Todas las instrucciones incluyen un codop principal de cuatro bits y una referencia a un registro de predicado. Si bien el campo codop principal solo puede distinguir 16 posibilidades, la interpretación de dicho campo depende del valor de plantilla y de la posición de la instrucción dentro de un paquete (Tabla 15.3),

Tabla 15.3. Codificación del campo plantilla y correspondencia del conjunto de instrucciones.

Plantilla	Instrucción 0	Instrucción 1	Instrucción 2
00	Unidad M	Unidad I	Unidad I
01	Unidad M	Unidad I	Unidad I
02	Unidad M	Unidad I	Unidad I
03	Unidad M	Unidad I	Unidad I
04	Unidad M	Unidad L	Unidad X
05	Unidad M	Unidad L	Unidad X
08	Unidad M	Unidad M	Unidad I
09	Unidad M	Unidad M	Unidad I
0A	Unidad M	Unidad M	Unidad I
0B	Unidad M	Unidad M	Unidad I
0C	Unidad M	Unidad F	Unidad I
0D	Unidad M	Unidad F	Unidad I
0E	Unidad M	Unidad M	Unidad F
0F	Unidad M	Unidad M	Unidad F
10	Unidad M	Unidad I	Unidad B
11	Unidad M	Unidad I	Unidad B
12	Unidad M	Unidad B	Unidad B
13	Unidad M	Unidad B	Unidad B
16	Unidad B	Unidad B	Unidad B
17	Unidad B	Unidad B	Unidad B
18	Unidad M	Unidad M	Unidad B
19	Unidad M	Unidad M	Unidad B
1C	Unidad M	Unidad F	Unidad B
1D	Unidad M	Unidad F	Unidad B

proporcionando por tanto más códigos de operación posibles. Las instrucciones típicas también incluyen tres campos para referenciar registros, dejando diez bits para otra información necesaria para especificar completamente la instrucción.

FORMATO DEL LENGUAJE ENSAMBLADOR

Como en cualquier conjunto de instrucciones máquina, se proporciona un lenguaje ensamblador para comodidad del programador. El ensamblador o compilador traduce cada instrucción de lenguaje ensamblador en una instrucción IA-64 de 41 bits. El formato general de una instrucción de lenguaje ensamblador es:

[hp] nemotécnico[.comp] dest = fuentes

donde

hp	Especifica un registro de predicado de un bit usado para habilitar la instrucción. Si el valor del registro es 1 (verdadero) en tiempo de ejecución, la instrucción se ejecuta y el resultado es entregado al hardware. Si el valor es falso, el resultado de la instrucción no se entrega sino que se descarta. La mayoría de las instrucciones IA-64 pueden habilitarse por un predicado, pero no obligatoriamente. Para representar una instrucción que no utilice predicado, el valor *hp* se pone a 0 y el registro de predicado cero siempre tiene un valor constante de 1.
nemotécnico	Especifica el nombre de una instrucción IA-64.
comp	Especifica uno o más complementos de instrucción, separados por puntos, que se usan para modificar el nemotécnico. No todas las instrucciones requieren el uso de un complemento.
dest	Especifica uno o más operandos destino, siendo habitual un único destino.
fuentes	Especifica uno o más operandos fuente. La mayoría de las instrucciones tienen dos o más operandos fuente.

En cada línea, todos los caracteres situados a la derecha de una barra doble "//" se consideran un comentario. Los grupos de instrucciones y las paradas se indican por medio de un punto y coma doble ";;". Un **grupo de instrucciones** se define como una secuencia de instrucciones que no tiene dependencias de los tipos lectura después de escritura o escritura después de escritura. El procesador puede emitir dichas instrucciones sin comprobaciones hardware sobre dependencias entre registros. He aquí un ejemplo sencillo:

```
ld8     r1 = [r5];;       // Primer grupo
add     r3 = r1, r4       // Segundo grupo
```

La primera instrucción lee un valor de ocho bytes desde la posición de memoria situada en la dirección indicada por el registro r5 y después sitúa el valor en el registro r1. La segunda instrucción suma el contenido de los registros r1 y 4 y coloca el resultado en r3. Dado que la segunda instrucción depende del valor del registro r1, que es alterado por la primera, las dos instrucciones no pueden estar en el mismo grupo de ejecución en paralelo.

A continuación se muestra un ejemplo más complejo, con varias dependencias en el flujo de registros:

```
ld8     r1 = [r5]          // Primer grupo
sub     r6 = r8, r9;;      // Primer grupo
add     r3 = r1, r4        // Segundo grupo
st8     [r6] = r12         // Segundo grupo
```

La última instrucción almacena el contenido de r12 en la posición de memoria cuya dirección está en r6.

Ya estamos listos para estudiar los cuatro mecanismos clave de la arquitectura IA-64 para sustentar el paralelismo en las instrucciones:

- Uso de predicados.
- Especulación en el control.
- Especulación en los datos.
- Segmentación software.

La Figura 15.3, basada en otra figura de [HALF97], ilustra las dos primeras técnicas, discutidas en esta subsección y en las siguientes.

EJECUCIÓN CON PREDICADOS

El uso de predicados es una técnica mediante la cual el procesador determina qué instrucciones pueden ejecutarse en paralelo. En este método, el compilador elimina saltos del programa usando ejecución condicional. Un ejemplo típico de un lenguaje de alto nivel es la instrucción **if-then-else**. Un compilador clásico inserta un salto condicional en el punto del **if** de esta construcción. Si la condición tiene cierto resultado lógico, el salto no se produce y se ejecuta el siguiente bloque de instrucciones, que representa el camino del **then**; al final de este camino hay un salto incondicional que evita el siguiente bloque, que representa el camino del **else**. Si la condición tiene el resultado lógico

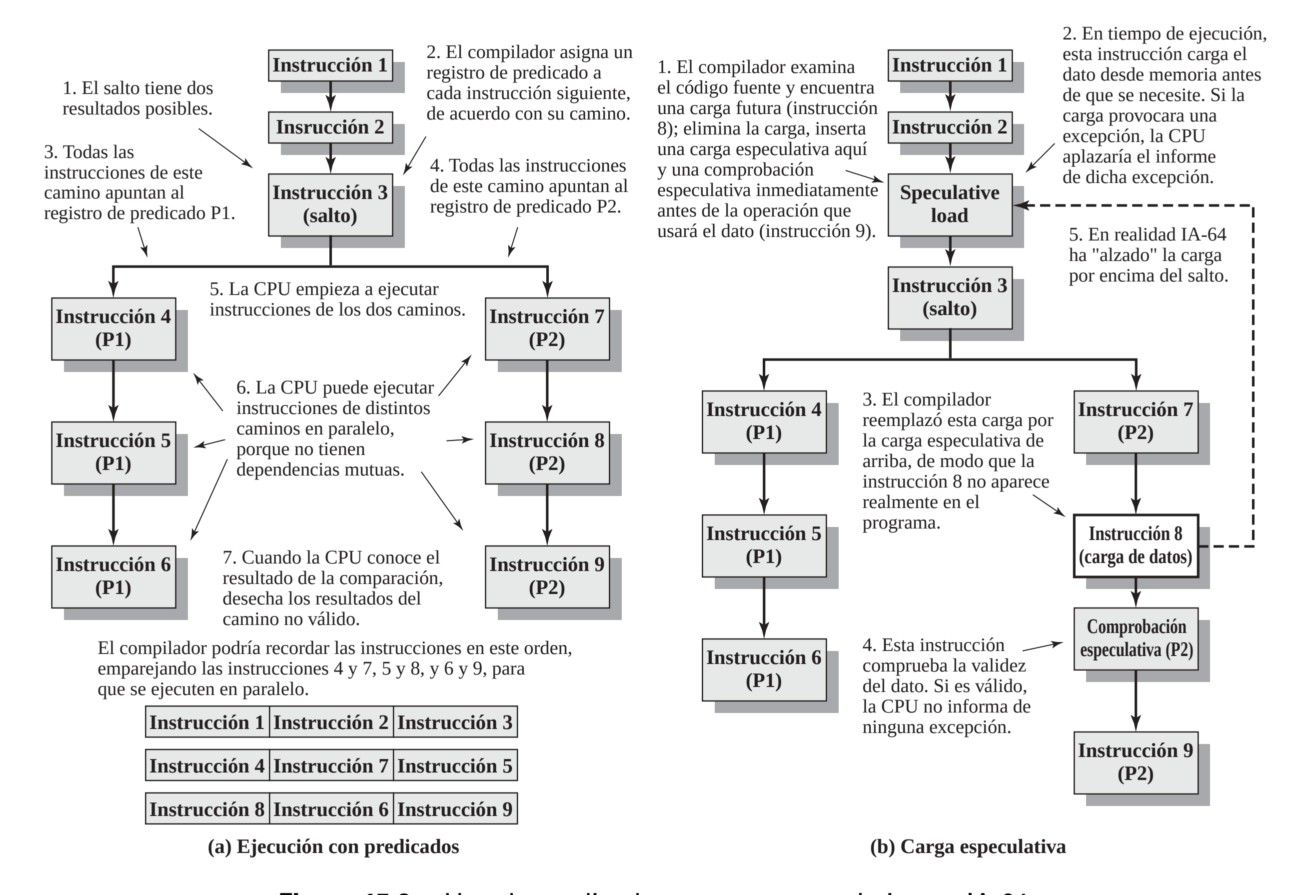


Figura 15.3. Uso de predicados y carga especulativa en IA-64.

contrario, se produce el salto, que evita el bloque de instrucciones del **then**, y la ejecución prosigue por el bloque de instrucciones del **else**. Los dos flujos de instrucciones se juntan tras el final del bloque del **else**. Un compilador de IA-64 en lugar de eso hace lo siguiente (Figura 15.3a):

1. En el punto del programa del **if**, inserta una instrucción de comparación que crea dos predicados. Si la comparación es verdadera, el primer predicado se ajusta a verdadero y el segundo a falso; si la comparación es falsa, el primer predicado se ajusta a falso y el segundo a verdadero.
2. Añade a cada instrucción del camino del **then** una referencia a un registro de predicado que contiene el valor del primer predicado, y añade a cada instrucción del camino del **else** una referencia a un registro de predicado que contiene el valor del segundo predicado.
3. El procesador ejecuta instrucciones de los dos caminos. Cuando el resultado de la comparación se conoce, el procesador desecha los resultados de un camino y entrega los resultados del otro camino. Esto permite al procesador introducir instrucciones de ambos caminos en el cauce de instrucciones sin esperar a que termine la operación de comparación.

A modo de ejemplo, considere el siguiente código fuente:

Código fuente:

```
if (a&&b)
     j = j + 1;
else
     if (c)
          k = k + 1;
else
          k = k – 1;
i = i + 1;
```

Dos sentencias **if** seleccionan uno de tres posibles caminos de ejecución. Esto puede compilarse en el siguiente código, usando el lenguaje ensamblador del Pentium. El programa tiene tres instrucciones de salto condicional y una de salto incondicional:

Código en ensamblador:

```
     cmp   a,   0    ; compara a con 0
     je    L1        ; salta a L1 si a = 0
     cmp   b,   0
     je    L1
     add   j,   1    ; j = j + 1
     jmp   L3
L1:  cmp   c,   0
     je    L2
     add   k,   1    ; k = k + 1
     jmp   L3
L2:  sub   k,   1    ; k = k – 1
L3:  add   i,   1    ; i = i + 1
```

En el lenguaje ensamblador del Pentium, un punto y coma se usa para delimitar un comentario.

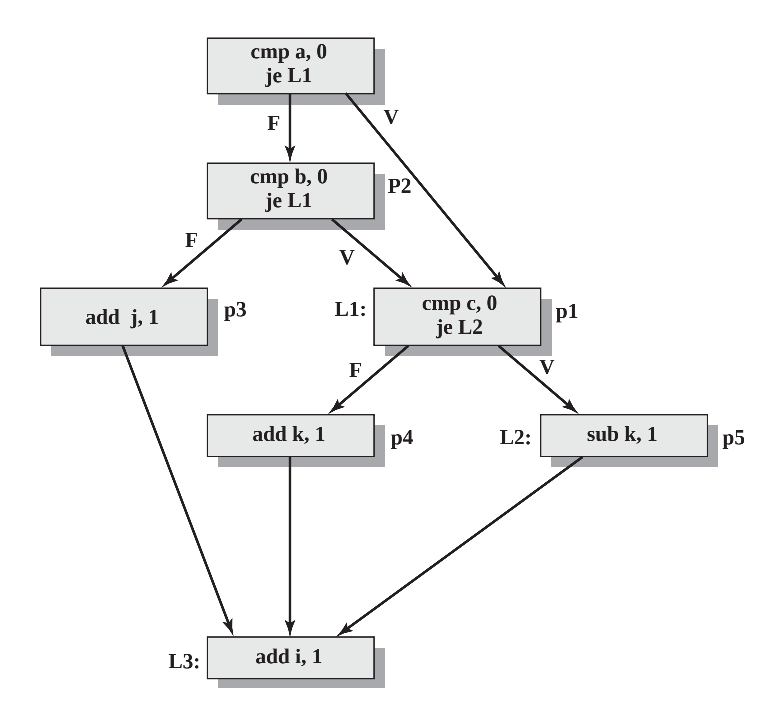


Figura 15.4. Ejemplo de uso de predicados (V = verdadero, F = falso).

La Figura 15.4 muestra un diagrama de flujo de este código en ensamblador. Este diagrama divide el programa en lenguaje ensamblador en bloques de código separados. El compilador puede asignar un predicado a cada bloque que se ejecuta condicionalmente. Los predicados se indican en la Figura 15.4. Suponiendo que a todos estos predicados se les haya asignado un valor inicial de falso, el código resultante en ensamblador de IA-64 queda como sigue:

Código con predicados:

```
(1)         cmp.eq p1,   p2 = 0, a;;
(2)  (p2)   cmp.eq p1,   p3 = 0, b
(3)  (p3)   add j = 1,   j
(4)  (p1)   cmp.ne p4,   p5 = 0, c
(5)  (p4)   add k = 1,   k
(6)  (p5)   add k = −1, k
(7)         add i = 1,   i
```

La instrucción (1) compara el contenido del registro simbólico con 0; pone el valor del registro de predicado p1 a 1 (verdadero) y p2 a 0 (falso) si la relación es verdadera, y pondrá el valor del predicado p1 a 0 y p2 a 1 si la relación es falsa. La instrucción (2) se ejecuta solo si el predicado p2 es verdadero (es decir, si a es verdadero, lo que es equivalente a decir que a $\neq$ 0). El procesador captará, decodificará, y comenzará a ejecutar esta instrucción, pero solo tomará la decisión de entregar el resultado una vez que determine si el valor del registro de predicado p1 es 1 o 0. Observe que la instrucción (2) es una instrucción generadora de predicado y ella misma constituye un predicado. Esta instrucción requiere tres campos de registro de predicado en su formato.

Volviendo a nuestro programa de Pentium, los dos primeros saltos condicionales en el código en ensamblador de Pentium se traducen a dos instrucciones de comparación generadoras de predicados en IA-64. Si la instrucción (1) pone p2 a falso, la instrucción (2) no se ejecutará. Tras la instrucción (2) en el programa IA-64, p3 es verdadero solo si la sentencia **if** externa del código fuente es verdadera. Esto es, el predicado p3 es verdadero solo si la expresión (a AND b) es verdadera (es decir, a $\neq$ 0 AND b $\neq$ 0). La parte del **then** de la sentencia **if** externa se asocia al predicado p3 por esta razón. La instrucción (4) del código IA-64 decide si se ejecuta la instrucción de suma o la de resta de la parte del **else** externo. Por último, el incremento de i se realiza incondicionalmente. Examinando el código fuente y después el código con predicados, vemos que se va a ejecutar solo una de las instrucciones (3), (5) y (6). En un procesador superescalar normal, usaríamos predicción de saltos para estimar cuál de las tres va a ejecutarse y seguir ese camino. Si el procesador hace una estimación equivocada, el cauce ha de vaciarse. Un procesador IA-64 puede comenzar la ejecución de las tres instrucciones y, una vez que se conozcan los valores de los registros de predicado, entregar solamente el resultado de la instrucción válida. Por tanto, utilizamos las unidades de ejecución en paralelo adicionales para evitar los retardos debidos al vaciado del cauce.

Gran parte de la investigación original sobre ejecución con predicados se hizo en la Universidad de Illinois. Los estudios de simulación indican que la utilización de predicados se traduce en una reducción sustancial de saltos dinámicos y fallos en la predicción de saltos, y en una mejora considerable de las prestaciones de procesadores con múltiples cauces paralelos (por ejemplo, [MAHL94], [MAHL95]).

ESPECULACIÓN EN EL CONTROL

Otra innovación importante de la arquitectura IA-64 es la especulación en el control, también conocida como carga especulativa. Permite al procesador cargar datos desde memoria antes de que el programa los necesite, para evitar retrasos por el acceso a memoria. Además, el procesador aplaza el informe de excepciones hasta que sea necesario comunicar la excepción. Se usa el término *alzar* («*hoist*») para hacer referencia al movimiento de una instrucción de carga a un punto anterior en el flujo de instrucciones.

La minimización de los retardos provocados por las cargas es crucial para mejorar las prestaciones. Típicamente, al principio de un bloque de código, hay varias operaciones de carga que llevan datos desde la memoria a los registros. Como la memoria, aunque se le hayan añadido uno o dos niveles de caché, es lenta comparada con el procesador, los retardos para obtener datos de la memoria se convierten en un cuello de botella. Para minimizarlo, interesaría reordenar el código de forma que las cargas se hicieran tan pronto como fuera posible. Esto puede hacerse con cualquier compilador, hasta cierto punto. El problema se presenta cuando intentamos mover una carga de una parte a otra en un flujo de control. No se puede mover incondicionalmente la carga más arriba de un salto porque la carga puede no producirse realmente. Podríamos mover la carga condicionalmente, usando predicados, de modo que los datos pudieran recogerse de la memoria pero no entregarse a un registro de la arquitectura mientras no se conociera el resultado del predicado; o podríamos usar técnicas de predicción de saltos del tipo que vimos en el Capítulo 14. El problema de esta estrategia es que la carga puede dilatarse. Podrá generarse una excepción debida a una dirección no válida o una falta de página. Si esto ocurre, el procesador tendría que ocuparse de la excepción o la falta, causando un retraso.

¿Cómo podemos, entonces, mover la carga más arriba del salto? La solución indicada en IA-64 es la carga especulativa, que separa el funcionamiento de la carga (entregar un valor) del de la excepción

(Figura 15.3b). Una instrucción de carga del programa original se reemplaza por dos instrucciones:

- Una carga especulativa (ld.s) ejecuta la captación desde memoria y lleva a cabo la detección de excepciones, pero no lanza la excepción (no llama a la rutina del SO que maneja la excepción). Esta instrucción ld.s se alza a un punto anterior del programa que sea adecuado.
- Una instrucción de comprobación (check.s) permanece en el lugar de la carga original y lanza las excepciones. Esta instrucción check.s puede asociarse a un predicado de forma que solo se ejecute si el predicado es verdadero.

Si ld.s detecta una excepción, ajusta un bit de recuerdo asociado con el registro destino, conocido como el bit *Not a Thing* (NaT)[1]. Si la instrucción check.s correspondiente se ejecuta, y el bit NaT está a uno, la instrucción check.s salta a una rutina de servicio de excepción.

Veamos un ejemplo sencillo, tomado de [INTE00a, Volumen 1]. Este es el programa original:

```
(p1)    br      alguna_etiqueta   // Ciclo 0
        ld8     r1 = [r5];;       // Ciclo 1
        add     r2 = r1, r3       // Ciclo 3
```

La primera instrucción salta si el predicado p1 es verdadero (el registro p1 tiene el valor 1). Observe que las instrucciones de salto y de carga están en el mismo grupo de instrucciones, a pesar de que la carga no debería ejecutarse si se produce el salto. IA-64 garantiza que si el salto se produce, las instrucciones posteriores, incluso las del mismo grupo de instrucciones no se ejecutarán. Las implementaciones de IA-64 pueden usar predicción de saltos para intentar mejorar la eficiencia, pero deben evitar resultados incorrectos. Por último, observe que la instrucción de suma se ve retrasada al menos un periodo de reloj (un ciclo) debido a la latencia de memoria de la operación de carga.

El compilador puede reescribir el código usando una carga especulativa y una comprobación:

```
        ld8.sr1 = [r5];;          // Ciclo –2
        // Otras instrucciones
(p1)    br      alguna_etiqueta   // Ciclo 0
        chk.s   r1, recuperacion  // Ciclo 0
        add     r2 = r1, r3       // Ciclo 0
```

No se puede mover sin más la instrucción de carga encima de la instrucción de salto dejándola como está, porque dicha instrucción puede provocar una excepción (por ejemplo, r5 puede contener un puntero nulo). En lugar de ello, se convierte la carga en una carga especulativa, ld8.s, y después se mueve. La carga especulativa no provoca inmediatamente una excepción cuando la detecta; solo registra ese hecho activando el bit NaT del registro destino (en este caso, r1). La carga especulativa se ejecuta ahora incondicionalmente al menos dos ciclos antes del salto. La instrucción chk.s comprueba si el bit NaT de r1 está activo. Si no lo está, la ejecución sencillamente prosigue por la siguiente instrucción. Si lo está, se produce un salto a un programa de recuperación. Observe que las

[1] N. del T.: *Not a Thing* (ninguna cosa) expresa el hecho de que el registro asociado no contiene un valor válido cuando este bit vale 1.

instrucciones de salto, comprobación y suma se han comentado como instrucciones ejecutadas en el mismo ciclo de reloj. Sin embargo, el hardware asegura que el resultado producido por la carga especulativa no actualiza el estado de la aplicación (no cambia los contenidos de r1 y r2) si no ocurren dos condiciones: el salto no se produce (p1 = 0), y la comprobación no detecta una excepción aplazada (r1.NaT = 0).

Hay un punto importante a destacar en este ejemplo. Si no hay excepción, la carga especulativa es una carga real y tiene lugar antes de la rama del programa que se supone que se va a seguir. Si el salto se produce, entonces habrá ocurrido una carga que no estaba contemplada en el programa original. El programa, tal como está escrito, supone que r1 no se lee en la rama de la bifurcación seguida cuando el salto tiene lugar. Si r1 es leído en esa rama, el compilador debe usar otro registro para albergar el resultado especulativo.

Veamos otro ejemplo más complejo, utilizado por Intel y HP para probar los programas con predicados y para ilustrar el uso de cargas especulativas, y conocido como el problema de las ocho reinas. El objetivo es colocar ocho reinas en un tablero de ajedrez de tal manera que ninguna reina amenace a otra. La Figura 15.5a muestra una solución. La línea principal del código fuente, en un bucle interno, es la siguiente:

```
if ((b[j] == true) && (a[i + j] == true) && (c[i − j] == true))
```

donde $1 \leq i, j \leq 8$.

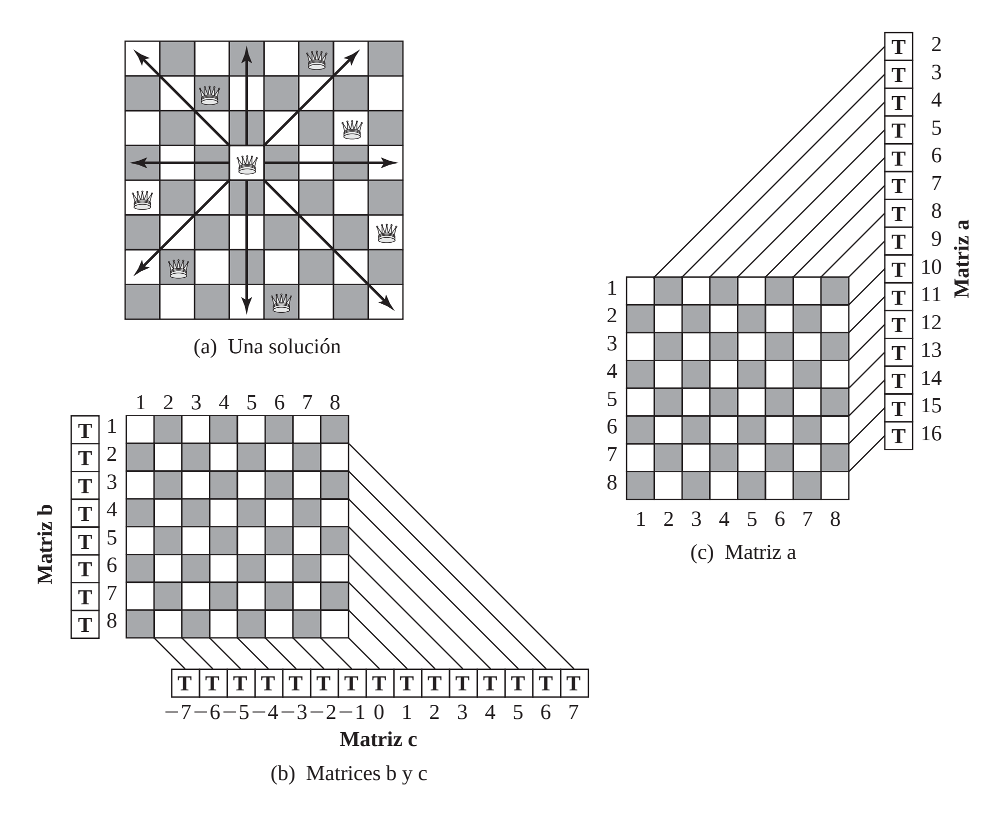


Figura 15.5. El problema de las ocho reinas.

El mecanismo de rastreo de conflictos entre reinas consiste en tres matrices booleanas que guardan el estado de las reinas en cada fila y en cada diagonal. TRUE significa que no hay ninguna reina en esa fila o diagonal; FALSE significa que ya hay una reina. Las Figuras 15.5b y c muestran la correspondencia de las matrices con el tablero de ajedrez. Todos los elementos de la matriz se inicializan a TRUE. Los elementos del 1 al 8 en la matriz B corresponden a las filas de la 1 a la 8 en el tablero. Si hay una reina en la fila *n*, se pone b[*n*] a FALSE. Los elementos de la matriz C están numerados desde −7 a 7 y corresponden a la diferencia entre los números de columna y de fila, que define las diagonales que van hacia abajo a la derecha. Si hay una reina en la columna 1, fila 1, se pone c[0] a FALSE. Si hay una reina en la columna 1, fila 8, se pone c[−7] a FALSE. Los elementos de la matriz A se numeran del 2 al 16 y corresponden a la suma de la columna y la fila. Una reina colocada en la columna 1, fila 1, hace que a[2] valga FALSE. Una reina colocada en la columna 3, fila 5, pone a[8] a FALSE.

El programa global se mueve por columnas, colocando una reina en cada columna de tal modo que no sea atacada por una reina colocada con anterioridad en su fila o en una de las dos diagonales.

Un programa simple en ensamblador del Pentium incluye tres cargas y tres saltos:

Código en ensamblador:

```
(1)           mov r2, &b[j]   ; transferir contenido
                              ; de la posición
                              ; b[j] al registro r2
(2)           cmp r2, 1
(3)           jne L2
(4)           mov r4, &a[i + j]
(5)           cmp R4, 1
(6)           jne L2
(7)           mov r6, &c[i − j]
(8)           cmp R6, 1
(9)           jne L2
(10)   L1:    <código del camino del then>
(11)   L2:    <código del camino del else>
```

En el programa precedente, la notación &x simboliza una dirección inmediata de la posición x.

Con cargas especulativas y ejecución con predicados queda lo siguiente:

Código con cargas especulativas y predicados:

```
(1)              mov r1 = &b[j]          // transferir dirección
                                         // de b[j] a r1
(2)              mov r3 = &a[i + j]
(3)              mov r5 = &c[i − j]
(4)              ld8 r2 = [r1]           // carga indirecta via r1
(5)              ld8.s r4 = [r3]
(6)              ld8.s r6 = [r5]
(7)              cmp.eq p1, p2 = 1, r2
(8)       (p2)   br L2
```

```
(9)           chk.s r4, recuperacion_a  // comprobar la
                                        // carga de a
(10)          cmp.eq p3, p4 = 1, r4
(11)    (p4)  br L2
(12)          chk.s r6, recuperacion_b  // comprobar la
                                        // carga de b
(13)          cmp.eq p5, p6 = 1, r5
(14)    (p6)  br L2
(15)  L1:     <código del camino del then>
(16)  L2:     <código del camino del else>
```

El programa en ensamblador se compone de tres bloques básicos de código, cada uno de los cuales es una carga seguida de un salto condicional. Las instrucciones de ajuste de dirección 4 y 7 del código en ensamblador del Pentium son cálculos aritméticos sencillos; pueden hacerse en cualquier momento, así que el compilador los coloca al principio. Después el compilador afronta los tres bloques simples, cada uno de los cuales consiste en una carga, un cálculo de condición, y un salto condicional. Parece haber poca esperanza de hacer algo en paralelo aquí. Además, si suponemos que la carga necesita dos o más ciclos de reloj, se pierde algún tiempo antes de que el salto condicional pueda ejecutarse. Lo que puede hacer el compilador es alzar la segunda y la tercera carga (instrucciones 5 y 8 en el código del Pentium) por encima de todos los saltos. Esto se hace poniendo una carga especulativa al principio (instrucciones IA-64 5 y 6) y dejando una comprobación en el bloque de código original (instrucciones IA-64 9 y 12).

Esta transformación hace posible ejecutar las tres cargas en paralelo y comenzar pronto las cargas a fin de minimizar o evitar los retrasos debidos al tiempo de carga. El compilador puede ir más lejos con un uso más agresivo de los predicados, y eliminar dos de los tres saltos:

Código con cargas especulativas y predicados revisado:

```
(1)           mov r1 = &b[j]
(2)           mov r3 = &a[i + j]
(3)           mov r5 = &c[i − j]
(4)           ld8 r2 = [r1]
(5)           ld8.s r4 = [r3]
(6)           ld8.s r6 = [r5]
(7)           cmp.eq p1, p2 = 1, r2
(8)     (p1)  chk.s r4, recuperacion_a
(9)     (p1)  cmp.eq p3, p4 = 1, r4
(10)    (p3)  chk.s r6, recuperacion_b
(11)    (p3)  cmp.eq p5, p6 = 1, r5
(12)    (p6)  br L2
(13)  L1:     <código del camino del then>
(14)  L2:     <código del camino del else>
```

Ya teníamos una comparación que generaba dos predicados. En el código revisado, en lugar de saltar en el predicado falso, el compilador habilita la ejecución de la comprobación y la siguiente

comparación en el predicado verdadero. Eliminar los dos saltos significa eliminar dos errores de predicción potenciales, de manera que el ahorro es mayor que solo dos instrucciones.

ESPECULACIÓN EN LOS DATOS

En una especulación en el control, se mueve una carga a una posición anterior dentro de una secuencia de código para compensar la latencia de carga, y se hace una comprobación para asegurar que no ocurra una excepción si posteriormente se comprueba que la carga no debía ejecutarse. En la especulación en los datos, se mueve una carga antes de una instrucción de almacenamiento que podría alterar la posición de memoria fuente de la carga. Se realiza una comprobación posterior para asegurar que la carga recibe el valor de memoria correcto. Para explicar el mecanismo, usaremos un ejemplo tomado de [INTE00a, Volumen 1].

Considere el siguiente fragmento de programa:

```
st8   [r4] = r12          // Ciclo 0
ld8   r6 = [r8];;         // Ciclo 0
add   r5 = r6, r7;;       // Ciclo 2
st8   [r18] = r5          // Ciclo 3
```

Tal como está escrito, el código requiere cuatro ciclos de instrucción para ejecutarse. Si los registros r4 y r8 no contienen la misma dirección de memoria, el almacenamiento a través de r4 no puede afectar al valor que hay en la dirección contenida en r8; bajo esta circunstancia, es seguro reordenar la carga y el almacenamiento para llevar más rápidamente el valor a r6, que se necesita después. No obstante, como las direcciones de r4 y r8 podrían coincidir o solaparse, tal intercambio no es seguro. IA-64 resuelve este problema con el uso de una técnica conocida como carga avanzada.

```
ld8.ar6 = [r8];;          // Ciclo —2 o anterior;
                          // carga avanzada
// otras instrucciones
st8   [r4] = r12          // Ciclo 0
ld8.cr6 = [r8]            // Ciclo 0; comprobar carga
add   r5 = r6, r7;;       // Ciclo 0
st8   [r18] = r5          // Ciclo 1
```

En este código hemos movido la instrucción ld a una posición anterior y la hemos convertido en una carga avanzada. Además de realizar la carga especificada, la instrucción ld8.a escribe su dirección fuente (la dirección contenida en r8) en una estructura de datos hardware conocida como tabla de direcciones de cargas avanzadas (*Advanced Load Address Table*, ALAT). Cada instrucción de almacenamiento IA-64 comprueba si la ALAT contiene elementos que se solapen con su dirección destino; si se encuentra una coincidencia, el elemento de la ALAT se elimina. Cuando la comprobación de la carga se ejecuta, comprueba si en la ALAT hay alguna dirección coincidente. Si encuentra alguna, significa que ninguna instrucción de almacenamiento entre la carga avanzada y la comprobación de la carga ha alterado la dirección fuente de la carga, y no se realiza ninguna acción. Sin embargo, si la instrucción de comprobación de la carga no encuentra ningún elemento coincidente en la ALAT, la operación de carga se realiza de nuevo para asegurar el resultado correcto.

También podríamos desear ejecutar especulativamente otras instrucciones que dependan de datos de una instrucción de carga junto con la propia carga. Partiendo del mismo programa original, supongamos que movemos hacia arriba la carga y la instrucción de suma posterior:

```
            ld8.ar6 = [r8];;        // Ciclo −3 o anterior;
                                    // carga avanzada
            // otras instrucciones
            add  r5 = r6, r7;;      // Ciclo −1; suma que usa r6
            // otras instrucciones
            st8  [r4] = r12         // Ciclo 0
            chk.a r6, recuperación  // Ciclo 0; comprobación
back:                               // punto de retorno del salto
                                    // a recuperación
            st8  [r18] = r5         // Ciclo 0
```

Aquí hemos usado una instrucción chk.a en lugar de una instrucción ld8.c para validar la carga avanzada. Si la instrucción chk.a determina que la carga ha fallado, no puede conformarse con volver a ejecutar la carga; por el contrario, salta a una rutina de recuperación que vuelve a ejecutar las instrucciones:

```
recuperación:
            ld8  r6 = [r8];;        // vuelve a cargar r6 de [r8]
            add  r5 = r6, r7;;      // vuelve a ejecutar la suma
            br   back               // vuelve al código principal
```

Esta técnica es eficaz solo si las cargas y almacenamientos implicados tienen poca probabilidad de solapamiento.

SEGMENTACIÓN SOFTWARE

Considere el siguiente bucle:

```
L1:     ld4   r4 = [r5], 4;;        // Ciclo 0; carga con postinc. 4
        add   r7 = r4, r9;;         // Ciclo 2
        st4   [r6] = r7, 4          // Ciclo 3; almac. con postinc. 4
        br.cloop L1;;               // Ciclo 3
```

Este bucle suma una constante a un vector y almacena el resultado en otro vector (es decir, y[i] = x[i] + c). La instrucción ld4 carga cuatro bytes desde memoria. El modificador «, 4» al final de la instrucción indica que se trata de la forma de la instrucción de carga que actualiza la base; la dirección que hay en r5 se incrementa en cuatro después de que tenga lugar la carga. De un modo similar, la instrucción st4 almacena cuatro bytes en memoria y la dirección que hay en r6 se incrementa en cuatro después del almacenamiento. La instrucción br.cloop, conocida como un salto a bucle contabilizado, usa el registro de aplicación contador de bucles (*Loop Count*, LC). Si el registro LC es mayor que cero, se decrementa y el salto tiene lugar. El valor inicial de LC es el número de iteraciones del bucle.

Observe que en este programa, virtualmente no hay ocasión para paralelismo en las instrucciones dentro del bucle. Además, las instrucciones de la iteración *x* se ejecutan antes de que comience la iteración $x + 1$. No obstante, si no hay conflicto de direcciones entre la carga y el almacenamiento (r5 y r6 apuntan a posiciones de memoria disjuntas), la utilización podría mejorar moviendo instrucciones independientes desde la iteración $x + 1$ a la iteración *x*. Otra forma de expresarlo es que si desenrollamos el código del bucle escribiendo explícitamente un nuevo conjunto de instrucciones para cada iteración, entonces sí hay una oportunidad para incrementar el paralelismo. Veamos qué podría hacerse con cinco iteraciones:

```
ld4   r32   = [r5],   4 ;;   // Ciclo 0
ld4   r33   = [r5],   4 ;;   // Ciclo 1
ld4   r34   = [r5],   4      // Ciclo 2
add   r36   = r32,   r9 ;;   // Ciclo 2
ld4   r35   = [r5],   4      // Ciclo 3
add   r37   = r33,   r9      // Ciclo 3
st4   [r6]  = r36,    4 ;;   // Ciclo 3
ld4   r36   = [r5],   4      // Ciclo 4
add   r38   = r34,   r9      // Ciclo 4
st4   [r6]  = r37,    4 ;;   // Ciclo 4
add   r39   = r35,   r9      // Ciclo 5
st4   [r6]  = r38,    4 ;;   // Ciclo 5
add   r40   = r36,   r9      // Ciclo 6
st4   [r6]  = r39,    4 ;;   // Ciclo 6
st4   [r6]  = r40,    4 ;;   // Ciclo 7
```

Este programa completa cinco iteraciones en siete ciclos, en comparación con los veinte ciclos del programa iterativo original. Se asume que hay dos puertos de acceso a memoria de manera que se puedan ejecutar en paralelo una carga y un almacenamiento. Este es un ejemplo de segmentación software, análoga a la segmentación hardware. La Figura 15.6 ilustra el proceso. El paralelismo se consigue agrupando instrucciones de distintas iteraciones. Para que esto funcione, los registros temporales usados dentro del bucle tienen que cambiarse en cada iteración para evitar conflictos entre registros. En este caso, se han usado dos registros temporales (r4 y r7 en el programa original). En el programa expandido, el número de registro de cada registro se incrementa en cada iteración, y los números de registro se inicializan suficientemente separados para evitar solapamientos.

La Figura 15.6 muestra que el cauce software tiene tres fases. Durante la **fase de prólogo**, se inicia una nueva iteración en cada ciclo de reloj y el cauce se llena gradualmente. Durante la **fase de núcleo**, el cauce está lleno, alcanzando un paralelismo máximo. En nuestro ejemplo, se ejecutan tres instrucciones en paralelo durante la fase de núcleo, aunque el ancho del cauce es cuatro. Durante la **fase de epílogo**, se completa una iteración en cada ciclo de reloj.

La segmentación software mediante desenrollamiento de bucles cede al compilador o al programador la responsabilidad de asignar correctamente los nombres de los registros. Además, en bucles largos con muchas iteraciones, el desenrollamiento origina un aumento significativo del tamaño del código. Para un bucle indeterminado (número total de iteraciones desconocido en tiempo de compilación), la tarea es aun más complicada por la necesidad de hacer un desenrollamiento parcial y de controlar después el contador del bucle. IA-64 proporciona soporte hardware para realizar la

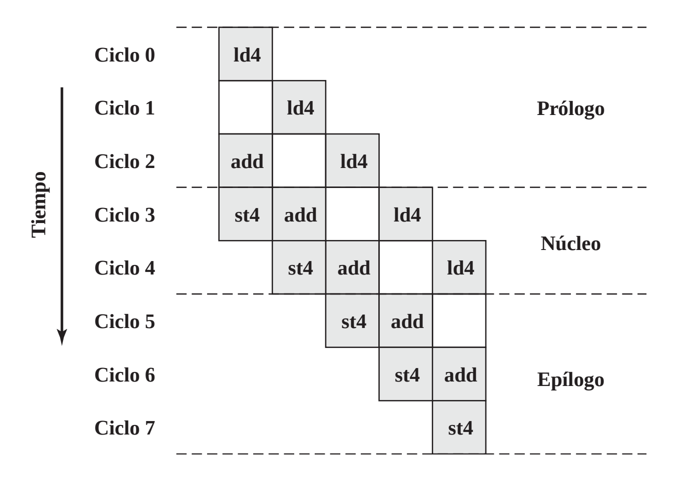


Figura 15.6. Ejemplo de segmentación sofware.

segmentación software sin expansión de código y con una mínima responsabilidad por parte del compilador. Las características clave para dar soporte a la segmentación software son:

- **Renombramiento de registros automático:** un área de tamaño fijo de los bancos de registros de predicados y de coma flotante (p16 a p63; fr32 a fr127) y un área de tamaño programable del banco de registros generales (rango máximo de r32 a r127) permiten rotación. Esto significa que durante cada iteración de un bucle con segmentación software, las referencias a registros dentro de esos rangos se incrementan automáticamente. Por tanto, si un bucle utiliza el registro r32 en la primera iteración, automáticamente utilizará r33 en la segunda iteración, y así sucesivamente.
- **Uso de predicados:** cada instrucción del bucle forma parte de un predicado con un registro de predicado rotatorio asociado. Esto tiene el propósito de determinar si el cauce está en la fase de prólogo, núcleo o epílogo, como se explica más adelante.
- **Instrucciones especiales de finalización del bucle:** son instrucciones de salto que hacen que los registros roten y que el número de iteraciones del bucle se decremente.

Este es un asunto relativamente complejo; presentamos aquí un ejemplo que ilustra algunas de las capacidades de segmentación software de IA-64. Tomamos el programa del bucle original de esta sección y mostramos cómo programarlo con segmentación por software, suponiendo un número de iteraciones de 200 y la existencia de dos puertos de acceso a memoria:

```
          mov lc = 199          // fijar el registro contador de bucle
                                // a 199,
                                // igual al número de iteraciones — 1
          mov ec = 4            // hacer registro contador de epílogo
                                // igual
                                // al número de etapas de epílogo + 1
          mov pr.rot = 1<<16;;  // pr16 = 1; rest = 0
L1: (p16) ld4 r32 = [r5], 4     // Ciclo 0
    (p17) ---                   // Etapa vacía
```

```
(p18)   add r35 = r34, r9       // Ciclo 0
(p19)   st4 [r6] = r36, 4       // Ciclo 0
        br.ctop L1;;            // Ciclo 0
```

Resumimos los puntos clave relacionados con este programa:

1. El cuerpo del bucle se divide en múltiples *etapas*, con cero o más instrucciones por etapa.

2. La ejecución del bucle tiene lugar a través de tres fases. Durante la fase de prólogo, una nueva iteración del bucle comienza cada vez, añadiendo una etapa al cauce. Durante la fase de núcleo, comienza una iteración del bucle y termina otra cada vez; el cauce está lleno, con el número máximo de etapas activas. Durante la fase de epílogo, no comienzan nuevas iteraciones y termina una iteración cada vez, vaciando el cauce software.

3. Se asigna un predicado a cada etapa para controlar la activación de las instrucciones de esa etapa. Durante la fase de prólogo, p16 es verdadero y p17, p18 y p19 son falsos en la primera iteración. En la segunda iteración, p16 y p17 son verdaderos; durante la tercera iteración, p16, p17 y p18 son verdaderos. Durante la fase de núcleo, todos los predicados son verdaderos. Durante la fase de epílogo, los predicados se vuelven falsos uno por uno, comenzando por p16. Los cambios en los valores de predicado se realizan por medio de la rotación de registros de predicado.

4. Todos los registros generales con números mayores que 31 rotan en cada iteración. Los registros rotan a números de registro mayores de una manera cíclica. Por ejemplo, el valor del registro x estará localizado en el registro $x + 1$ después de una rotación; esto no se consigue moviendo los valores sino mediante el renombramiento de los registros por hardware. Por consiguiente, en nuestro ejemplo, el valor que la carga escribe en el registro r32 es leído por la suma dos iteraciones (y dos rotaciones) después como r34. Del mismo modo, el valor que la suma escribe en r35 es leído por el almacenamiento una instrucción después como r36.

5. En la instrucción br.ctop, el salto se produce si $LC > 0$ o si $EC > 1$. La ejecución de br.ctop tiene los efectos adicionales siguientes. Si $LC > 0$, entonces LC se decrementa; esto ocurre durante las fases de prólogo y de núcleo. Si $LC = 0$ y $EC > 1$, EC se decrementa; esto ocurre durante la fase de epílogo. La instrucción también controla la rotación de registros. Si $LC > 0$, cada ejecución de br.ctop pone un 1 en p63. Con la rotación, p63 se convierte en p16, creando una secuencia continua de unos en los registros de predicado durante las fases de prólogo y de núcleo. Si $LC = 0$, br.ctop pone p63 a 0, introduciendo ceros en los registros de predicado durante la fase de epílogo.

La Tabla 15.4 muestra una traza de ejecución de este ejemplo.

15.4. ARQUITECTURA DE CONJUNTO DE INSTRUCCIONES IA-64

La Figura 15.7 muestra el conjunto de registros disponible para los programas de aplicación. Es decir, estos registros son visibles por las aplicaciones y pueden leerse y, en la mayoría de los casos, escribirse. Los conjuntos de registros incluyen:

- **Registros generales:** 128 registros de uso general de 64 bits. Asociado a cada registro hay un bit NaT usado para rastrear las excepciones especulativas aplazadas, tal como se explicó en la

Tabla 15.4. Traza del bucle del ejemplo de segmentación software.

Ciclo	Unidad de ejecución/Instrucción				Estado antes de br.ctop					
	M	I	M	B	p16	p17	p18	p19	LC	EC
0	ld4			br.ctop	1	0	0	0	199	4
1	ld4			br.ctop	1	1	0	0	198	4
2	ld4	add		br.ctop	1	1	1	0	197	4
3	ld4	add	st4	br.ctop	1	1	1	1	196	4
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
100	ld4	add	st4	br.ctop	1	1	1	1	99	4
. . .	. . .. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
199	ld4	add	st4	br.ctop	1	1	1	1	0	4
200		add	st4	br.ctop	0	1	1	1	0	3
201		add	st4	br.ctop	0	0	1	1	0	2
202			st4	br.ctop	0	0	0	1	0	1
					0	0	0	0	0	0

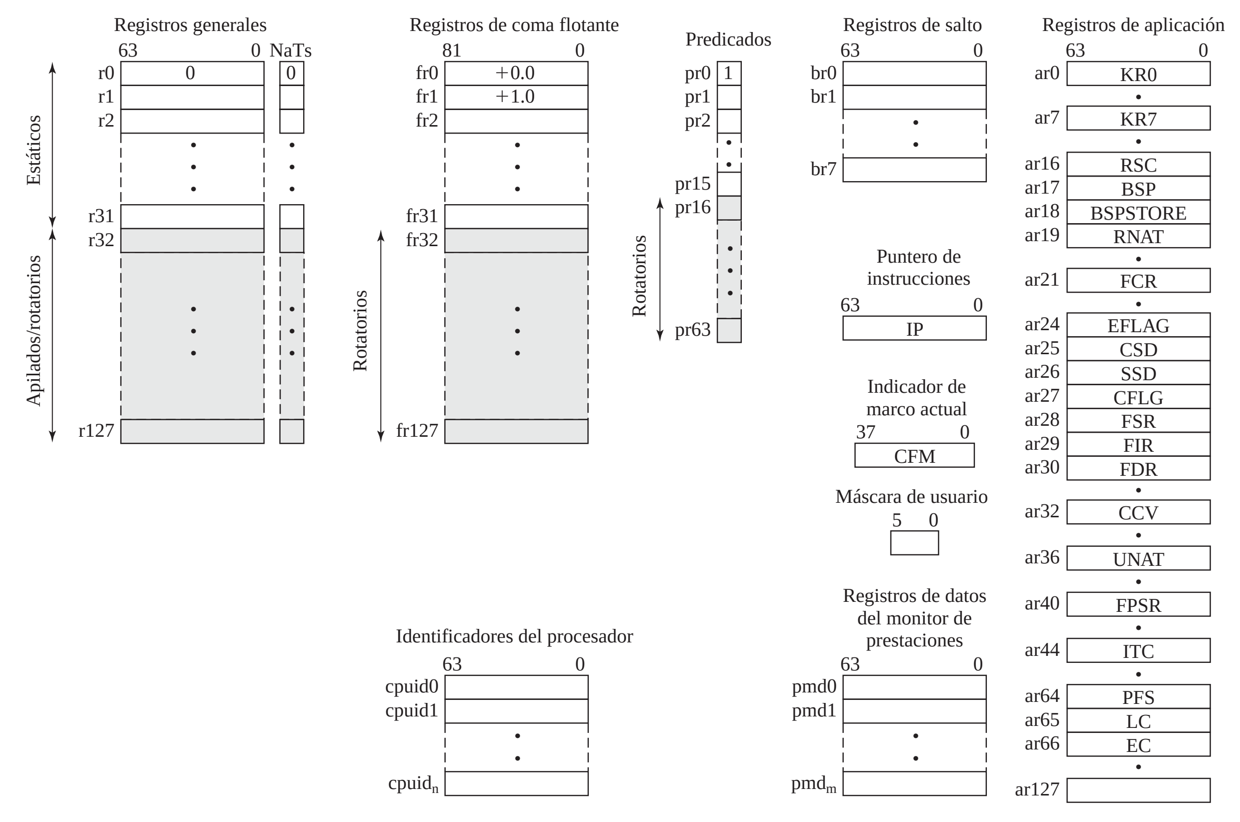


Figura 15.7. Conjunto de registros de aplicación de IA-64.

Sección 15.3. A los registros r0 hasta r31 se les conoce como registros estáticos; una referencia de un programa a cualquiera de ellos se interpreta literalmente. Los registros r32 hasta r127 pueden usarse como registros rotatorios en la segmentación software (discutida en la Sección 15.3) y para la implementación de la pila de registros (discutida más adelante en esta sección). Las referencias a estos registros son virtuales, y el hardware puede realizar renombramiento de registros dinámicamente.

- **Registros de coma flotante:** 128 registros de 82 bits para números en coma flotante. Este tamaño es suficiente para manejar números en el formato doble extendido de IEEE 754 (*ver* Tabla 9.3). Los registros fr0 hasta fr31 son estáticos, y los registros fr32 hasta fr127 pueden usarse como registros rotatorios en la segmentación software.
- **Registros de predicado:** 64 registros de 1 bit usados como predicados. El registro pr0 siempre vale uno para habilitar instrucciones no asociadas a predicados. Los registros pr0 hasta pr15 son estáticos, y los registros pr16 hasta pr63 pueden usarse como registros rotatorios en la segmentación software.
- **Registros de salto:** ocho registros de 64 bits usados para saltos.
- **Puntero de instrucciones (*Instruction Pointer,* IP):** guarda la dirección del paquete de la instrucción IA-64 que se ejecuta actualmente.
- **Indicador de marco actual (*Current frame marker,* CFM):** guarda información de estado relacionada con el marco de pila de registros generales en curso e información de rotación para los registros fr y pr.
- **Máscara de usuario:** un conjunto de valores de un solo bit usado para excepciones de alineación, monitores de prestaciones, y para monitorizar el uso de los registros de coma flotante.
- **Registros de datos del monitor de prestaciones:** usados como apoyo al hardware de monitorización de prestaciones.
- **Identificadores del procesador:** describen características del procesador dependientes de la implementación.
- **Registros de aplicación:** una colección de registros de uso específico. La Tabla 15.5 ofrece una breve descripción de cada uno.

PILA DE REGISTROS

El mecanismo de pila de registros de IA-64 evita el movimiento innecesario de datos hacia y desde los registros en las llamadas y retornos de procedimientos. El mecanismo proporciona a cada procedimiento invocado un nuevo **marco** de hasta 96 registros (r32 a r127) cuando se entra a dicho procedimiento. El compilador especifica el número de registros que necesita el procedimiento con la instrucción alloc, que indica cuántos registros son locales (usados solo dentro del procedimiento) y cuántos son salidas (usados para pasar parámetros a otro procedimiento llamado por este). Cuando tiene lugar una llamada a un procedimiento, el hardware IA-64 renombra los registros de manera que los registros locales del marco previo se ocultan, y los registros de salida del procedimiento que realiza la llamada pasan a tener números de registro comenzando por r32 en el procedimiento llamado. Los registros físicos dentro el rango r32 a r127 se asignan a los registros virtuales asociados con los procedimientos siguiendo una técnica de buffer circular. Es decir, el siguiente registro asignado después

Tabla 15.5. Registros de aplicación de IA-64.

Registros de núcleo (*kernel registers,* KR0-7)	Transmiten información desde el sistema operativo a la aplicación.
Configuración de la pila de registros (*Register Stack Configuration,* RSC)	Controla el funcionamiento del motor de la pila de registros (*Register Stack Engine,* RSE).
Puntero del almacén de salvaguarda del RSE (*RSE Backing Store Pointer,* BSP)	Contiene la dirección de memoria de la posición de salvaguarda de r32 en el marco de pila actual.
Puntero del almacén de salvaguarda del RSE para almacenamientos en memoria (*RSE Backing Store Pointer for memory stores,* BSPSTORE)	Contiene la dirección de memoria en la cual volcará el siguiente valor el RSE.
Registro de colección de NaT del RSE (*RSE NaT collection register,* RNAT)	Usado por el RSE para almacenar temporalmente los bits NaT cuando está volcando registros generales.
Valor de comparación e intercambio (*Compare and exchange Value,* CCV)	Contiene el valor de comparación usado como tercer operando fuente en la instrucción cmpxchg.
Registro de colección de NaT del usuario (*User NaT collection register,* UNAT)	Usado para almacenar temporalmente los bits NaT cuando se guardan y se restauran los registros generales en las instrucciones ld8.fill y st8.spill.
Registro de estado de coma flotante (*Floating-Point Status Register,* FPSR)	Controla excepciones, modo de redondeo, control de precisión, indicadores de estado y otros bits de control de las instrucciones de coma flotante.
Contador de tiempo de intervalos (*Interval Time Counter,* ITC)	Cuenta a una frecuencia proporcional a la frecuencia de reloj del procesador.
Estado de la función previa (*Previous Function State*, PFS)	Guarda el valor del registro CFM e información relacionada.
Contador de bucle (*Loop Count,* LC)	Usado en bucles contabilizados y decrementado por los saltos de tipo bucle-contabilizado.
Contador de epílogo (*Epilog Count,* EC)	Usado para contar el estado final (epílogo) en los bucles planificados modularmente.

de r127 es r32. Cuando es necesario, el hardware mueve los contenidos de los registros a memoria para liberar registros adicionales cuando tiene lugar una llamada a procedimiento, y restablece dichos contenidos desde memoria hacia los registros cuando se produce un retorno de procedimiento.

La Figura 15.8 ilustra el comportamiento de la pila de registros. La instrucción alloc incluye dos operandos, sof (*size of frame*, tamaño de marco) y sol (*size of locals*, tamaño de locales), para especificar el número necesario de registros. Dichos valores se almacenan en el registro CFM. Cuando se produce una llamada, los valores sol y sof del CFM se almacenan en los campos sol y sof del registro de aplicación estado de la función previa (*previous function state*, PFS) (Figura 15.9). En el retorno, estos valores sol y sof deben restaurarse desde el PFS al CFM. Para permitir llamadas y retornos anidados, los valores previos de los campos PFS deben salvaguardarse a través de las llamadas sucesivas de manera que puedan restaurarse en los retornos. De ello se encarga la instrucción alloc, que designa un registro general en el que se guarda el valor actual de los campos del PFS antes de que se sobrescriban desde los campos del CFM.

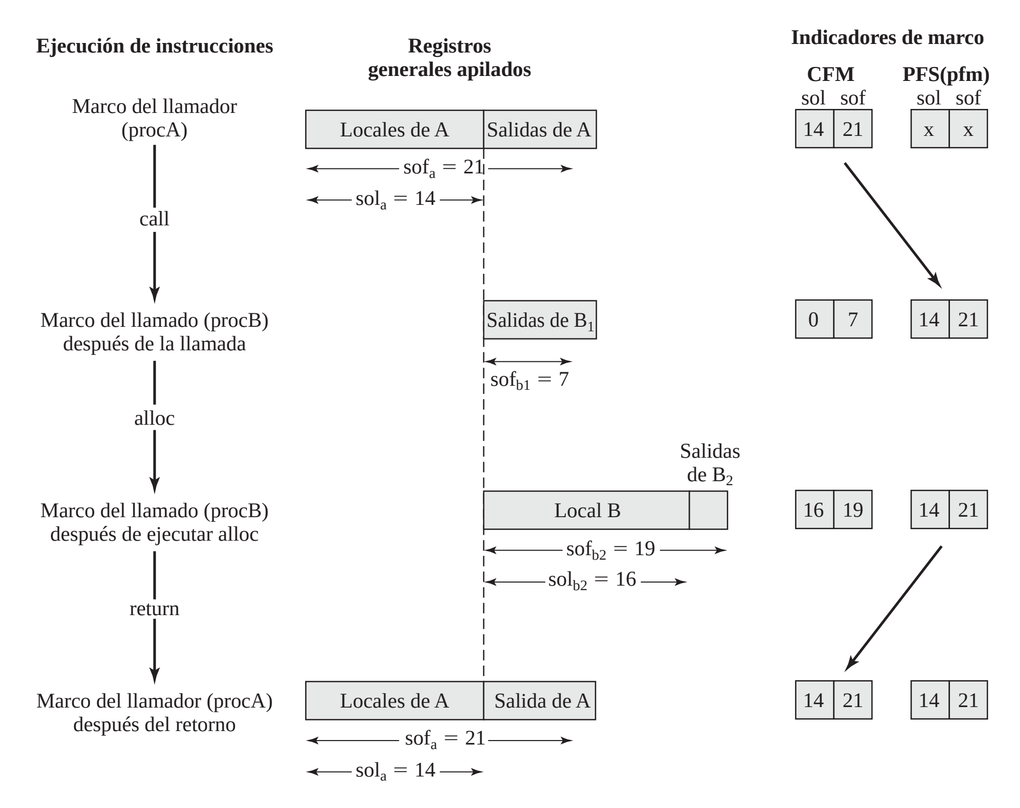


Figura 15.8. Comportamiento de la pila de registros en las llamadas y retornos de procedimientos.

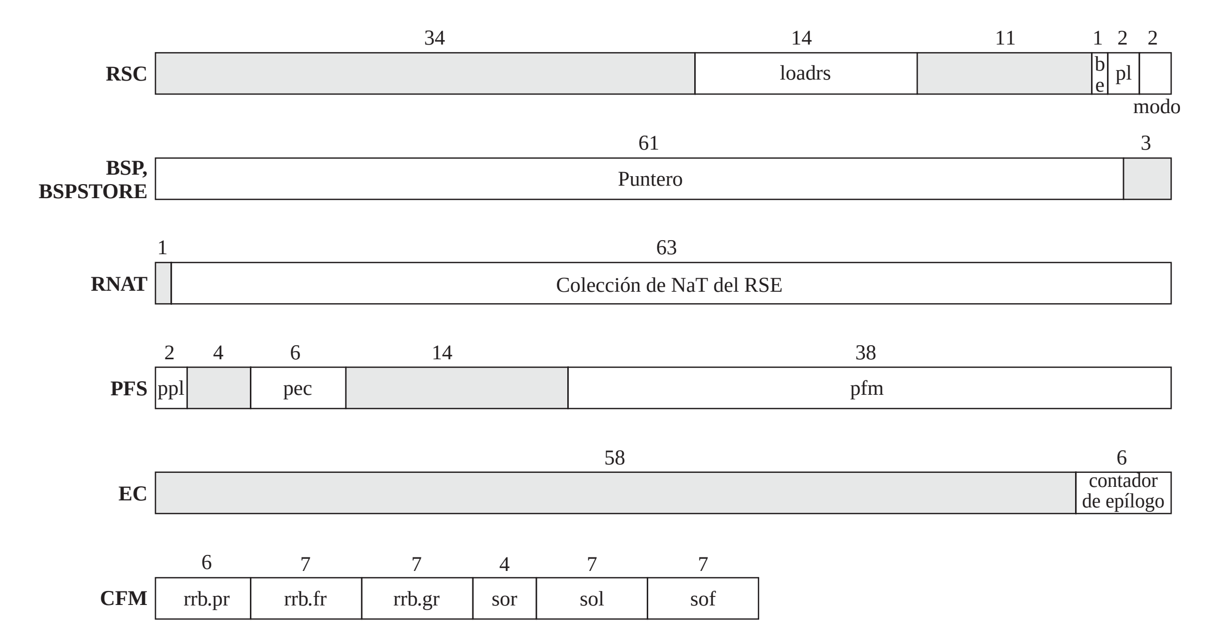


Figura 15.9. Formatos de algunos registros de IA-64.

INDICADOR DE MARCO ACTUAL Y ESTADO DE LA FUNCIÓN PREVIA

El registro CFM describe el estado del marco de pila de registros generales en curso, asociado al procedimiento activo actualmente. Incluye los siguientes campos:

- **sof:** tamaño del marco de pila.
- **sol:** tamaño de la porción de registros locales del marco de pila.
- **sor:** tamaño de la porción rotatoria del marco de pila; es un subconjunto de la porción local dedicado a la segmentación software.
- **valores base de renombramiento de registros:** valores usados en la ejecución de la rotación de registros generales, de coma flotante y de predicado.

El registro de aplicación PFS contiene los siguientes campos:

- **pfm:** indicador de marco previo; contiene todos los campos del CFM.
- **pec:** contador de epílogo previo.
- **ppl:** nivel de privilegio previo.

15.5. ORGANIZACIÓN DEL ITANIUM

El procesador Itanium de Intel es la primera implementación de la arquitectura de conjunto de instrucciones IA-64. La primera versión de esta implementación, conocida como Itanium, se lanzó en 2001 y fue seguida en 2002 por el Itanium 2. La organización del Itanium combina características superescalares con el soporte de las características exclusivas de IA-64 relacionadas con EPIC. Entre las características superescalares están el cauce segmentado hardware de anchura seis y de diez etapas, la precaptación dinámica, la predicción de saltos, y un marcador de registros para optimizar el no determinismo del tiempo de compilación. El hardware relacionado con EPIC incluye soporte para la ejecución con predicados, la especulación en el control y en los datos, y la segmentación software.

La Figura 15.10 es un diagrama de bloques general de la organización del Itanium. El Itanium incluye nueve unidades de ejecución: dos de enteros, dos de coma flotante, cuatro de memoria, y tres unidades de ejecución de saltos. Las instrucciones se captan de una caché de instrucciones L1 y pasan a un buffer que almacena hasta ocho paquetes de instrucciones. A la hora de decidir las unidades funcionales a donde se van a distribuir las instrucciones, el procesador ve como mucho dos paquetes de instrucciones a la vez. El procesador puede emitir un máximo de seis instrucciones por ciclo de reloj.

La organización es más sencilla en algunos aspectos que la de un superescalar convencional contemporáneo. El Itanium no usa centrales de reservas, ni buffers de reordenación, ni buffers de ordenación de memoria, ya que todos ellos han sido reemplazados por un hardware más sencillo dedicado a la especulación. El hardware de reasignación de registros es más simple que el renombramiento de registros típico de máquinas superescalares. La lógica de detección de dependencias entre registros no existe; ha sido reemplazada por las directivas de paralelismo explícito precalculadas por el software.

Mediante la predicción de saltos, el motor de captación/precaptación puede cargar especulativamente una instrucción de la caché L1 para minimizar los fallos de caché en la captación de

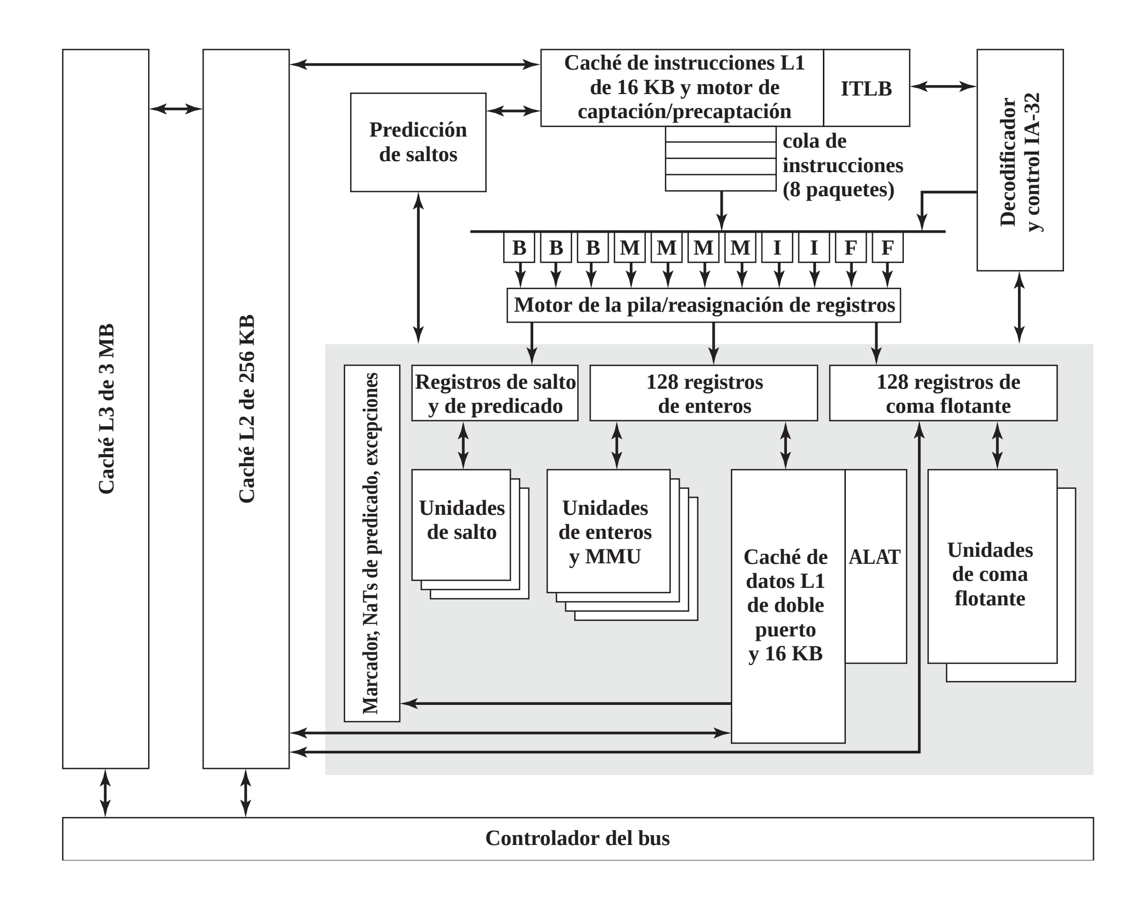


Figura 15.10. Organización del proceso Itanium 2.

instrucciones. El código captado se introduce en un buffer de separación que puede albergar hasta ocho paquetes de código.

Se usan tres niveles de caché. La caché L1 está dividida en una caché de instrucciones de 16 KB y una caché de datos de 16 KB, ambas asociativas por conjuntos de cuatro vías con un tamaño de línea de 32 bytes. La caché L2 de 256 KB es asociativa por conjuntos de seis vías con un tamaño de línea de 64 bytes. La caché L3 de 3 MB es asociativa por conjuntos de cuatro vías con un tamaño de línea de 64 bytes. Los tres niveles de caché están en el mismo chip que el procesador en el caso del Itanium 2. En el Itanium original, la caché L3 está fuera del chip pero en el mismo encapsulado que el procesador.

El Itanium 2 usa un cauce segmentado de ocho etapas para todas las instrucciones excepto las de coma flotante. La Figura 5.11 ilustra la relación entre las etapas del cauce y la organización del Itanium 2. Las etapas del cauce son:

- **Generación del puntero de instrucciones (*Instruction Pointer Generation,* IPG):** entrega un puntero de instrucciones a la caché L1I.
- **Rotación de instrucciones (ROT):** capta instrucciones y las cambia de posición de manera que el paquete 0 contenga la primera instrucción de deba ejecutarse.

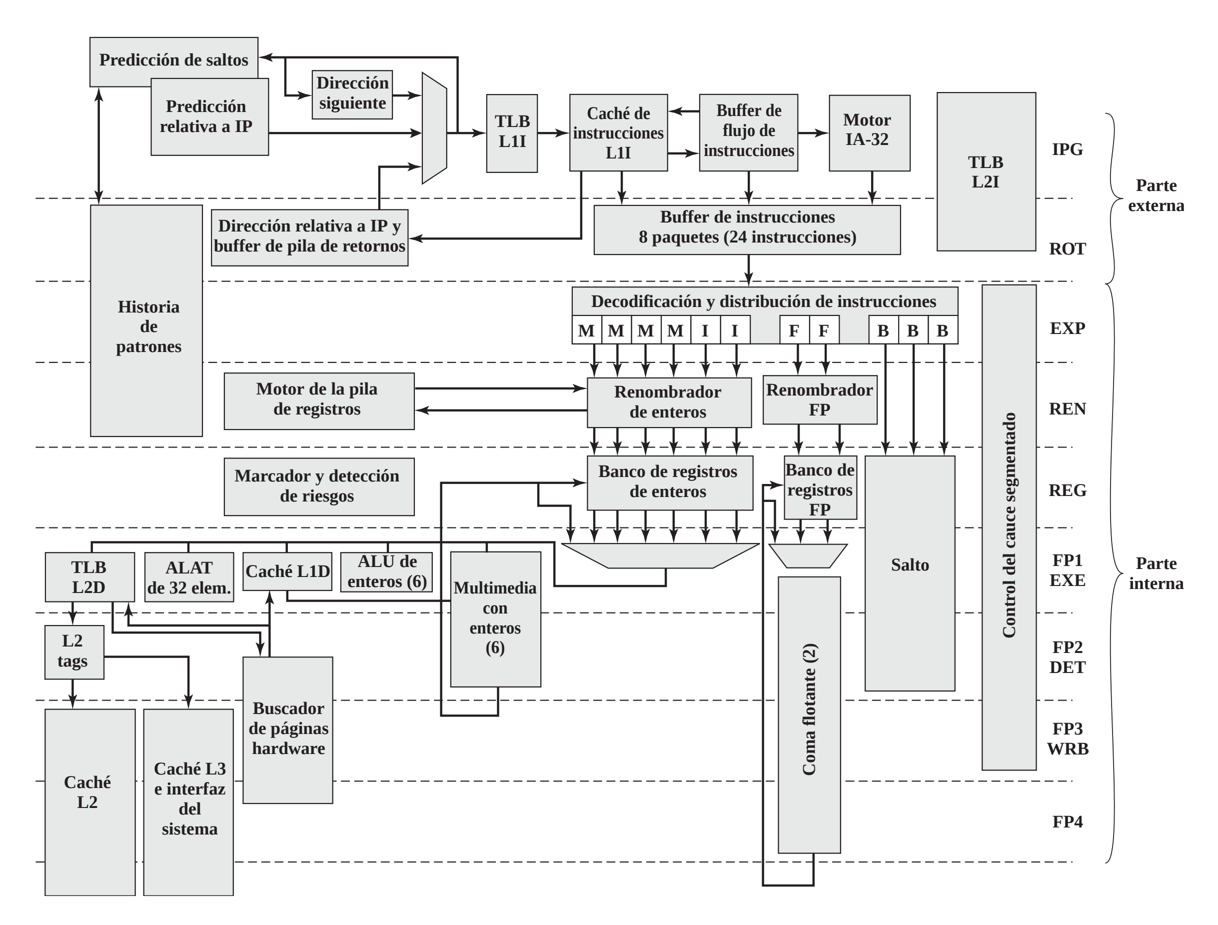


Figura 15.11. Cauce segmentado del procesador Itanium 2 [MCNA03].

- **Decodificación de plantillas, expansión y distribución de instrucciones (EXP):** decodifica plantillas de instrucciones, y distribuye hasta seis instrucciones a las unidades de ejecución a través de once puertos en conjunción con la información de los códigos de operación.
- **Renombramiento y decodificación (REN):** renombra (reorganiza) registros para el motor de pila de registros; decodifica instrucciones.
- **Lectura del banco de registros (REG):** entrega operandos a las unidades de ejecución.
- **Ejecución en la ALU (EXE):** ejecuta operaciones.
- **Última etapa de la detección de excepciones (DET):** detecta excepciones; abandona el resultado de la ejecución si el predicado de la instrucción no era verdadero; redirige saltos mal predichos.
- **Escritura (*Write back*, WRB):** escribe los resultados en el banco de registros.

En las instrucciones de coma flotante, las cinco primeras etapas del cauce segmentado son las mismas que acaban de enumerarse, seguidas de cuatro etapas de coma flotante, y seguidas a su vez de una etapa de escritura.

15.6. LECTURAS Y SITIOS WEB RECOMENDADOS

[HUCK00] ofrece una visión de conjunto de IA-64; otra visión general es [DULO98]. [SCHL00a] aporta una discusión sobre EPIC; un tratamiento más completo se ofrece en [SCHL00b]. Otros dos buenos estudios son [HWU01] y [KATH01]. [CHAS00] y [HWU98] ofrecen introducciones a la ejecución con predicados. El volumen 1 de [INTE00a] contiene un tratamiento detallado de la segmentación software; dos artículos que proporcionan una buena explicación de este tema, con ejemplos, son [JARP01] y [BHAR00].

Para una visión general de la arquitectura del procesador Itanium, vea [SHAR00]; [INTE00b] ofrece un tratamiento más detallado. [MCNA03] y [NAFF02] describen el Itanium 2 con cierto detalle.

[EVAN03], [TRIE01] y [MARK00] contienen más detalles sobre los temas de este capítulo. Por último, para una revisión exhaustiva de la arquitectura IA-64 y su conjunto de instrucciones, *vea* [INTE00a].

BHAR00 Bharandwaj, J. *et al.*: «The Intel IA-64 Compiler Code Generator». *IEEE Micro,* septiembre/octubre 2000.

CHAS00 Chasin, A.: «Predication, Speculation, and Modern CPUs.» *Dr. Dobb's Journal,* mayo, 2000.

DULO98 Dulong, C.: «The IA-64 Architecture at Work». *Computer,* julio, 1998.

EVAN03 Evans, J. y Trimper, G.: *Itanium Architecture for Programmers,* Upper Saddle River, NJ. Prentice Hall, 2003.

HUCK00 Huck, J. *et al.*: «Introducing the IA-64 Architecture». *IEEE Micro,* septiembre/octubre, 2000.

HWU98 Hwu, W.: «Introduction to Predicated Execution». *Computer,* enero, 1998.

HWU01 Hwu, W.; August, D. y Sias, J.: «Program Decision Logic Optimization Using Predication and Control Speculation». *Proceedings of the IEEE,* noviembre, 2001.

INTE00a Intel Corp. *Intel IA-64 Architecture Software Developer's Manual (4 volúmenes),* Documento 245317 hasta 245320. Aurora, CO, 2000.

INTE00b Intel Corp. *Itanium Processor Microarchitecture Reference for Software Optimization,* Aurora, CO, Documento 245473. Agosto, 2000.

JARP01 Jarp, S.: «Optimizing IA-64 Performance». *Dr. Dobb's Journal,* julio, 2001.

KATH01 Kathail, B.; Schlansker, M. y Rau, B.: «Compiling for EPIC Architectures». *Proceedings of the IEEE,* noviembre, 2001.

MARK00 Markstein, P.: *IA-64 and Elementary Functions,* Upper Saddle River, NJ. Prentice Hall PTR, 2000.

MCNA03 McNairy, C. y Soltis, D.: «Itanium 2 Processor Microarchitecture». *IEEE Micro,* marzo-abril, 2003.

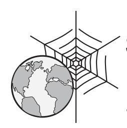
NAFF02 Naffziger, S. *et al.*: «The Implementation of the Itanium 2 Microprocessor». *IEEE Journal of Solid-State Circuits,* noviembre, 2002.

SCHL00a Schlansker, M. y Rau, B.: «EPIC: Explicitly Parallel Instruction Computing». *Computer,* febrero, 2000.

SCHL00b Schlansker, M. y Rau, B.: *EPIC: An Architecture for Instruction-Level Parallel Processors,* HPL Technical Report HPL-1999-111, Hewlett-Packard Laboratories **(www.hpl.hp.com),** febrero, 2000.

SHAR00 Sharangpani, H. y Arona, K.: «Itanium Processor Microarchitecture». *IEEE Micro,* septiembre/octubre, 2000.

TRIE01 Triebel, W.: *Itanium Architecture for Software Developers,* Intel Press, 2001.



SITIOS WEB RECOMENDADOS

- **Itanium:** sitio de Intel con la última información sobre IA-64 e Itanium.
- **Sitio de HP sobre la tecnología Itanium:** buena fuente de información.
- **IMPACT:** este es un sitio de la Universidad de Illinois, donde se ha hecho gran parte de la investigación sobre ejecución con predicados. Hay varios artículos disponibles sobre la materia.

15.7. PALABRAS CLAVE, PREGUNTAS DE REPASO Y PROBLEMAS

PALABRAS CLAVE

alzar	computación con instrucciones explícitamente paralelas (EPIC) especulación en el control	paquete
arquitectura IA-64	especulación en los datos	parada
bit NaT	grupo de instrucciones	pila de registros
campo plantilla	Itanium	registro de predicado
carga avanzada	marco de pila	saltos con predicados
carga especulativa	palabra de instrucción muy larga (VLIW)	segmentación software
codop principal		sílaba
complemento de instrucción		unidad de ejecución
		uso de predicados

PREGUNTAS DE REPASO

15.1. ¿Cuáles son los distintos tipos de unidades de ejecución en IA-64?
15.2. Explique el uso del campo plantilla en un paquete de IA-64.
15.3. ¿Cuál es el significado de una parada en el flujo de instrucciones?
15.4. Defina el uso de predicados y la ejecución con predicados.
15.5. ¿Cómo pueden reemplazar los predicados a una instrucción de salto condicional?
15.6. Defina especulación en el control.
15.7. ¿Cuál es el propósito del bit NaT?
15.8. Defina especulación en los datos.
15.9. ¿Cuál es la diferencia entre segmentación hardware y segmentación software?
15.10. ¿Cuál es la diferencia entre registros apilados y registros rotatorios?

PROBLEMAS

15.1. Suponga que un codop de IA-64 acepta tres registros como operandos y produce un registro como resultado. ¿Cuál es el número máximo de operaciones distintas que pueden definirse en una familia de codop principal?

15.2. ¿Cuál es el máximo número efectivo de codops principales?

15.3. En cierto punto de un programa de IA-64 hay diez instrucciones de tipo A y seis instrucciones de coma flotante que pueden emitirse concurrentemente. ¿Cuántas sílabas pueden aparecer sin ninguna parada entre ellas?

15.4. En el Problema 15.3:

(a) ¿Cuántos ciclos se necesitarían para una pequeña implementación de IA-64 que tuviera una unidad de coma flotante, dos unidades de enteros y dos unidades de memoria?

(b) ¿Cuántos ciclos se necesitan para la organización del Itanium de la Figura 15.10?

15.5. La implementación inicial del Itanium tenía dos unidades M y dos unidades I. ¿Cuáles de las plantillas de la Tabla 15.3 no pueden emparejarse como dos paquetes de instrucciones que puedan ejecutarse completamente en paralelo?

15.6. Un algoritmo que puede utilizar cuatro instrucciones de coma flotante por ciclo se codifica para IA-64. ¿Los grupos de instrucciones deben contener cuatro operaciones de coma flotante? ¿Cuáles son las consecuencias si la máquina en la que se ejecuta el programa tiene menos de cuatro unidades de coma flotante?

15.7. En la Sección 15.3, introdujimos las siguientes construcciones para la ejecución con predicados:

```
        cmp.crel p2,  p3 = a, b
(p1)    cmp.crel p2,  p3 = a, b
```

donde crel es una relación, tal como eq, ne, etc.; p1, p2 y p3 son registros de predicado; a es un registro o un operando inmediato; y b es un operando registro.

Complete la siguiente tabla de verdad:

p1	comparación	p2	p3
no presente	0		
no presente	1		
0	0		
0	1		
1	0		
1	1		

15.8. Para el programa con predicados de la Sección 15.3, que implementa el diagrama de flujo de la Figura 15.4, indique:

(a) Las instrucciones que pueden ejecutarse en paralelo.

(b) Las instrucciones que pueden agruparse en el mismo paquete de instrucciones de IA-64.

15.9. La arquitectura IA-64 incluye un conjunto de instrucciones multimedia comparables a las de la arquitectura IA-32 del Pentium (Tabla 10.11). Una instrucción de ese tipo es la instrucción de comparación paralela de la forma pcmp1, pcmp2, o pcmp4, que realiza una comparación paralela de 1, 2, o 4 bytes de una vez. La instrucción pcmp1.gt ri = rj, rk compara los dos operandos fuente (rj, rk) byte a byte. Para cada byte, si el byte en rj es mayor que el byte en rk, el correspondiente byte en ri se fijará a todo unos; en otro caso, el byte destino se rellenará con ceros. Ambos operandos se interpretan como operandos con signo.

Suponga que los registros r14 y r15 contienen las cadenas ASCII (ver Tabla 7.1) “00000000” y “99999999”, respectivamente, y que el registro r16 contiene una cadena arbitraria de ocho caracteres. Determine si los comentarios del siguiente fragmento de código son adecuados.

```
pcmp1.gt    r8 = r14,r16       // si algún carácter < "0" o
pcmp1.gt    r9 = r16,r15;;     // si algún carácter > "9"
cmp.ne      p6,p0 = r8,r0;;    // p6 = verdadero o
cmp.ne      p7,p0 = r9,r0;;    // p7 = verdadero de forma que
```

```
(p6) br error                        // este salto se ejecuta o
(p7) br error ;;                     // este salto se ejecuta
```

15.10. Considere el siguiente segmento de código fuente:

```
for (i = 0; i < 100; i++)
    if (A[i] > 50)
       j = j + 1;
    else
       k = k + 1;
```

(a) Escriba el correspondiente segmento de código en ensamblador del Pentium.

(b) Vuelva a escribir el segmento de código en ensamblador de IA-64 usando técnicas de ejecución con predicados.

15.11. Considere el siguiente fragmento de programa en C que utiliza valores en coma flotante:

```
a[i] = p * q;
c = a[j];
```

El compilador no puede establecer que $i \neq j$, pero tiene razones para creer que probablemente son distintos.

(a) Escriba un programa de IA-64 usando una carga avanzada para implementar este programa en C. *Ayuda:* las instrucciones de carga y de multiplicación en coma flotante son ldf y fmpy, respectivamente.

(b) Vuelva a escribir el programa usando predicados en lugar de la carga avanzada.

(c) ¿Cuáles son las ventajas e inconvenientes de las dos soluciones comparadas entre sí?

15.12. Suponga que se crea un marco de pila de registros con un tamaño SOF = 48. Si el tamaño del grupo de registros locales es SOL = 16:

(a) ¿Cuántos registros de salida (SOO) hay?

(b) ¿Qué registros están en los grupos de registros locales y de salida?

PARTE 4

LA UNIDAD DE CONTROL

CUESTIONES A TRATAR EN LA PARTE CUATRO

En la Parte Tres nos concentramos en las instrucciones máquina y las operaciones que lleva a cabo el procesador para ejecutar cada instrucción. Lo que quedó fuera de la discusión es qué se hace exactamente para que tenga lugar cada operación individual. Esa es la tarea de la unidad de control.

La unidad de control es la parte del procesador que realmente hace que ocurra todo. La unidad de control emite señales de control externas al procesador para producir el intercambio de datos con la memoria y los módulos de E/S. También emite señales de control internas al procesador para transferir datos entre registros, hacer que la ALU ejecute una función concreta, y regular otras operaciones internas. La entrada a la unidad de control está compuesta por el registro de instrucción, los indicadores de estado, y ciertas señales de control de fuentes externas (por ejemplo., señales de interrupción).

ESQUEMA DE LA PARTE CUATRO

CAPÍTULO 16. FUNCIONAMIENTO DE LA UNIDAD DE CONTROL

En el Capítulo 16 pasamos a una discusión sobre cómo se realizan las funciones del procesador o, más concretamente, sobre cómo se controlan los diversos elementos del procesador para proporcionar tales funciones, por medio de la unidad de control. Se muestra cómo cada ciclo de instrucción está compuesto de un conjunto de microoperaciones que generan señales de control. La ejecución se lleva a cabo por el efecto de esas señales de control, enviadas desde la unidad de control hacia la ALU, los registros y la estructura de interconexión del sistema. Por último, se presenta una solución a la implementación de la unidad de control, conocida como implementación cableada.

CAPÍTULO 17. CONTROL MICROPROGRAMADO

En el Capítulo 17, veremos cómo el concepto de microoperación conduce a un método elegante y potente para implementar la unidad de control, conocido como microprogramación. Fundamentalmente, se desarrolla un lenguaje de programación de un nivel más bajo. Cada instrucción del lenguaje máquina del procesador se traduce a una secuencia de instrucciones de la unidad de control. Estas instrucciones de menor nivel se llaman microinstrucciones, y el proceso de traducción es conocido como microprogramación. El capítulo describe el diseño de una memoria de control que contiene un microprograma para cada instrucción máquina. La estructura y la función de la unidad de control microprogramada se explican a partir de ese diseño.

CAPÍTULO 16

Funcionamiento de la unidad de control

PUNTOS CLAVE

- La ejecución de una instrucción implica la ejecución de una secuencia de pasos más pequeños, normalmente llamados ciclos. Por ejemplo, una ejecución puede constar de ciclos de captación, acceso indirecto a memoria, ejecución e interrupción. Además, cada ciclo se compone de una serie de operaciones más elementales, llamadas microoperaciones. Una única microoperación por lo general implica una transferencia entre registros, una transferencia entre un registro y un bus externo, o una operación sencilla de la ALU.
- La unidad de control de un procesador realiza dos tareas: (1) hace que el procesador ejecute las microoperaciones en la secuencia correcta, determinada por el programa que se está ejecutando, y (2) genera las señales de control que provocan la ejecución de cada microoperación.
- Las señales de control generadas por la unidad de control causan la apertura y el cierre de ciertas puertas lógicas, lo que da como resultado una transferencia de datos hacia, o desde, los registros, y una operación de la ALU.
- Una técnica para construir la unidad de control es la implementación cableada, en la cual dicha unidad es un circuito combinacional. Sus señales lógicas de entrada, gobernadas por la instrucción máquina en curso, se transforman en un conjunto de señales de control de salida.

En el Capítulo 10, señalamos que el conjunto de instrucciones máquina contribuye en gran medida a definir el procesador. Si conocemos el conjunto de instrucciones máquina, lo que incluye una comprensión del efecto de cada código de operación y de los modos de direccionamiento, y si conocemos el conjunto de registros visibles por el usuario, entonces conocemos las funciones que puede realizar el procesador. Esta no es una descripción completa. Necesitamos conocer las interfaces externas, por lo general accesibles a través de un bus, y cómo se manejan las interrupciones. Siguiendo esta línea de razonamiento, surge la siguiente lista de conceptos necesarios para especificar la funcionalidad de un procesador:

1. Operaciones (códigos de operación).
2. Modos de direccionamiento.
3. Registros.
4. Interfaz con el módulo de E/S.
5. Interfaz con el módulo de memoria.
6. Estructura del procesamiento de interrupciones.

Esta lista, aunque general, es bastante completa. Los puntos del 1 al 3 quedan definidos por el conjunto de instrucciones. Los puntos 4 y 5 vienen determinados típicamente por el bus del sistema. El punto 6 está definido parcialmente por el bus del sistema y parcialmente por el tipo de apoyo que ofrece el procesador al sistema operativo.

Los seis puntos de esta lista podrían denominarse requisitos funcionales de un procesador. Ellos determinan lo que debe hacer el procesador. Nos ocupamos de esto en las Partes Dos y Tres. Ahora nos vamos a centrar en la cuestión de cómo se realizan esas funciones o, más específicamente, cómo se controlan los diversos elementos del procesador para proporcionar esas funciones. Por tanto, vamos a estudiar la unidad de control, que controla el funcionamiento del procesador.

16.1. MICROOPERACIONES

Hemos visto que el funcionamiento de un computador, cuando ejecuta un programa, consiste en una secuencia de ciclos de instrucción, con una instrucción máquina por ciclo. Naturalmente, debemos recordar que esta secuencia de ciclos de instrucción no es necesariamente la misma que la *secuencia escrita* de instrucciones que constituye un programa, debido a la existencia de instrucciones de salto. A lo que nos referimos aquí es a la *secuencia temporal* de ejecución de instrucciones.

Hemos visto además que cada ciclo de instrucción puede considerarse compuesto por varias unidades más pequeñas. Una subdivisión práctica es captación, ciclo indirecto, ejecución e interrupción, si bien solo aparecen siempre los ciclos de captación y de ejecución.

Para diseñar una unidad de control, no obstante, necesitamos desglosar más esta descripción. En nuestra discusión sobre la segmentación en el Capítulo 12, comenzamos a ver que es posible una mayor descomposición. En realidad, veremos que cada uno de los ciclos más pequeños implica una serie de pasos, cada uno de los cuales involucra ciertos registros del procesador. Nos referiremos a estos pasos como *microoperaciones*. El prefijo *micro* alude al hecho de que cada paso es muy sencillo y hace muy poco. La Figura 16.1 representa la relación entre los distintos conceptos de los que hemos hablado. De forma resumida, la ejecución de un programa consiste en la ejecución secuencial de instrucciones. Cada instrucción se ejecuta durante un ciclo de instrucción compuesto por subciclos más cortos (por ejemplo, subciclo de captación, indirecto, de ejecución, de interrupción, etc.). La ejecución de cada subciclo incluye una o más operaciones más breves, es decir, una o más microoperaciones.

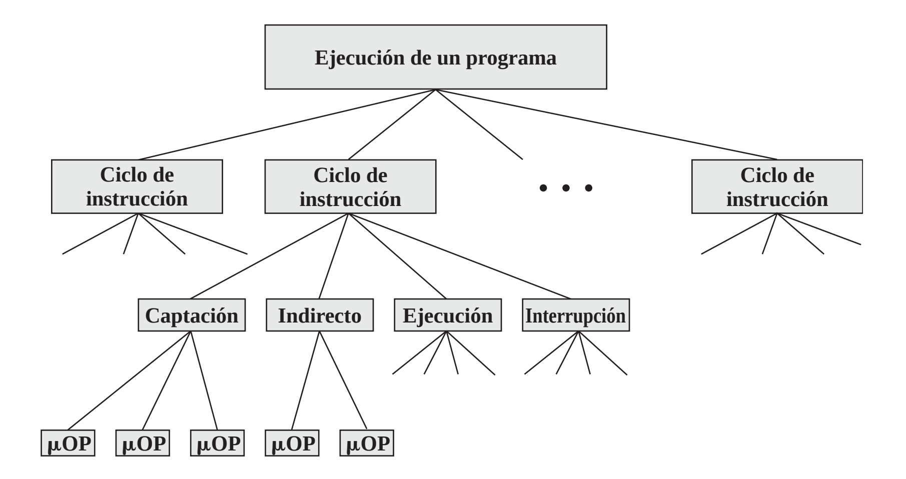


Figura 16.1. Elementos que constituyen la ejecución de un programa.

Las microoperaciones son la operaciones funcionales, o atómicas, de un procesador. En esta sección examinaremos las microoperaciones para llegar a comprender cómo los eventos de cualquier instrucción se pueden describir como una secuencia de tales microoperaciones. Usaremos un ejemplo sencillo. En el resto de este capítulo, mostraremos cómo el concepto de microoperaciones sirve como guía para el diseño de la unidad de control.

EL CICLO DE CAPTACIÓN

Comenzamos examinado el ciclo de captación, que tiene lugar al principio de cada ciclo de instrucción y hace que una instrucción sea captada de la memoria. Para el fin de este estudio, suponemos la organización representada en la Figura 12.6. Hay cuatro registros implicados:

- **Registro de dirección de memoria (*Memory Address Register,* MAR):** está conectado a las líneas de dirección del bus del sistema. Especifica la dirección de memoria de una operación de lectura o de escritura.
- **Registro intermedio de memoria (*Memory Buffer Register,* MBR):** está conectado a las líneas de datos del bus del sistema. Contiene el valor a almacenar en memoria o el último valor leído de memoria.
- **Contador de programa (*Program Counter,* PC):** contiene la dirección de la siguiente instrucción a captar.
- **Registro de instrucción (*Instruction Register,* IR):** contiene la última instrucción captada.

Consideremos la secuencia de eventos del ciclo de captación desde el punto de vista de su efecto sobre los registros del procesador. En la Figura 16.2 se muestra un ejemplo. Al comienzo del ciclo de captación, la dirección de la siguiente instrucción a ejecutar está en el contador de programa (PC); en este caso, la dirección es 1100100. El primer paso es llevar esa dirección al registro de dirección de

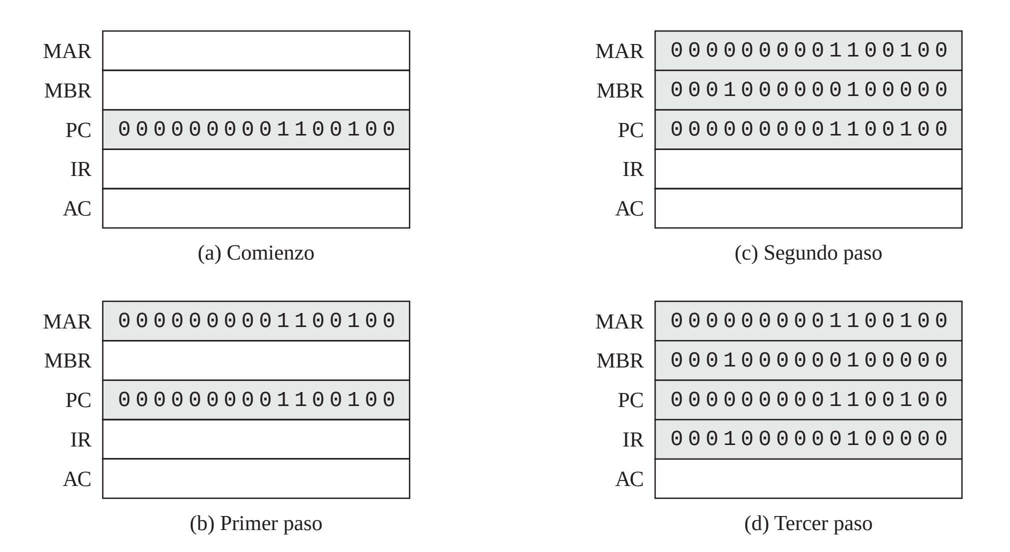


Figura 16.2. Secuencia de eventos del ciclo de captación.

memoria (MAR), ya que este es el único registro conectado a las líneas de dirección del bus del sistema. El segundo paso es traer la instrucción. La dirección deseada (en MAR) se coloca en el bus de direcciones, la unidad de control emite una orden READ por el bus de control, y el resultado aparece en el bus de datos y se copia en el registro intermedio de memoria (MBR). Es necesario además incrementar PC en *I* (longitud de la instrucción) para que esté preparado para la siguiente instrucción. Como estas dos acciones (leer una palabra de memoria, sumar *I* a PC) no se interfieren entre sí, podemos hacerlas simultáneamente para ahorrar tiempo. El tercer paso es transferir el contenido de MBR al registro de instrucción (IR). Esto libera MBR para su uso durante un posible ciclo indirecto.

De este modo, el sencillo ciclo de captación consta realmente de tres pasos y cuatro microoperaciones. Cada microoperación implica la transferencia de datos hacia dentro o hacia fuera de un registro. Con tal de que estas transferencias no se interfieran entre sí, varias de ellas pueden tener lugar durante un paso, ahorrando tiempo. Simbólicamente, podemos escribir esta secuencia de eventos como sigue:

$$
\begin{array}{ll}
t_1: & \text{MAR} \leftarrow (\text{PC}) \\
t_2: & \text{MBR} \leftarrow \text{Memoria} \\
 & \text{PC} \leftarrow (\text{PC}) + I \\
t_3: & \text{IR} \leftarrow (\text{MBR})
\end{array}
$$

donde *I* es la longitud de la instrucción. Tenemos que hacer varios comentarios sobre esta secuencia. Suponemos que se dispone de un reloj a efectos de temporización, y que este emite pulsos de reloj espaciados regularmente. Cada pulso de reloj define una unidad de tiempo. Así, todas las unidades de tiempo tienen la misma duración. Cada microoperación puede llevarse a cabo dentro de una única unidad de tiempo. La notación (t_1, t_2, t_3) representa las sucesivas unidades de tiempo. En palabras, tenemos:

- **Primera unidad de tiempo:** transferir el contenido de PC a MAR.
- **Segunda unidad de tiempo:** transferir el contenido de la posición de memoria especificada por MAR a MBR. Incrementar en *I* el contenido de PC.
- **Tercera unidad de tiempo:** transferir el contenido de MBR a IR.

Observe que las microoperaciones segunda y tercera tienen lugar durante la segunda unidad de tiempo. La tercera microoperación podría haberse agrupado con la cuarta sin afectar al funcionamiento de la captación:

$$
\begin{array}{ll}
t_1: & \text{MAR} \leftarrow (\text{PC}) \\
t_2: & \text{MBR} \leftarrow \text{Memoria} \\
t_3: & \text{PC} \leftarrow (\text{PC}) + I \\
 & \text{IR} \leftarrow (\text{MBR})
\end{array}
$$

Los agrupamientos de microoperaciones deben cumplir dos sencillas reglas:

1. Debe seguirse la secuencia correcta de eventos. Así, (MAR ← (PC)) debe preceder a (MBR ← Memoria), ya que la operación de lectura de memoria hace uso de la dirección almacenada en MAR.

2. Deben evitarse los conflictos. No se debe intentar leer y escribir en el mismo registro en una unidad de tiempo, ya que los resultados serían imprevisibles. Por ejemplo, las microoperaciones (MBR ← Memoria) e (IR ← MBR) no deberían tener lugar en la misma unidad de tiempo.

Un punto final digno de atención es que una de las microoperaciones incluye una suma. Para evitar la duplicación de circuitería, la suma podría realizarse en la ALU. El uso de la ALU puede implicar microoperaciones adicionales, dependiendo de la funcionalidad de la ALU y de la organización del procesador. Aplazamos la discusión de este punto hasta más adelante en este capítulo.

Es útil comparar los eventos descritos en esta y en las siguientes subsecciones con la Figura 3.5. Mientras que las microoperaciones se ignoraban en aquella figura, la presente discusión muestra las microoperaciones necesarias para llevar a cabo los subciclos del ciclo de instrucción.

EL CICLO INDIRECTO

Una vez que se capta una instrucción, el siguiente paso es captar los operandos fuente. Siguiendo con nuestro ejemplo sencillo, supongamos un formato de instrucción de una dirección, que permite direccionamiento directo e indirecto. Si la instrucción especifica una dirección indirecta, un ciclo indirecto ha de preceder al ciclo de ejecución. El flujo de datos difiere un poco del que se indicó en la Figura 12.27, e incluye las siguientes microoperaciones:

t_1: MAR ← (IR (dirección))
t_2: MBR ← Memoria
t_3: IR (dirección) ← (MBR (dirección))

El campo de dirección en la instrucción se transfiere a MAR. Este se usa después para captar la dirección del operando. Por último, el campo de dirección de IR se actualiza con el contenido de MBR, de modo que contenga una dirección directa en lugar de una indirecta.

IR tiene ahora el mismo estado que si no se hubiera usado direccionamiento indirecto, y está listo para el ciclo de ejecución. De momento saltamos ese ciclo, para considerar el ciclo de interrupción.

EL CICLO DE INTERRUPCIÓN

Cuando termina el ciclo de ejecución, se realiza una comprobación para determinar si ha ocurrido alguna interrupción habilitada. Si es así, tiene lugar un ciclo de interrupción. La naturaleza de este ciclo varía mucho de una máquina a otra. Aquí presentamos una secuencia muy simple de eventos, ilustrados en la Figura 12.8. Tenemos:

t_1: MBR ← (PC)
t_2: MAR ← Dirección de salvaguardia
PC ← Dirección de rutina
t_3: Memoria ← (MBR)

En el primer paso, el contenido de PC se transfiere a MBR, de modo que pueda guardarse para el retorno de la interrupción. Entonces MAR se carga con la dirección en la cual va a guardarse el

contenido de PC, y PC se carga con la dirección de comienzo de la rutina de procesamiento de la interrupción. Cada una de estas dos acciones puede ser una única microoperación. Sin embargo, ya que la mayoría de los procesadores tienen múltiples tipos y/o niveles de interrupciones, podrían hacer falta una o más microoperaciones adicionales para obtener la dirección de salvaguardia y la dirección de la rutina antes de que puedan transferirse a MAR y a PC, respectivamente. En todo caso, una vez hecho esto, el paso final es almacenar MBR, que contiene el antiguo valor de PC, en la memoria. El procesador queda entonces preparado para iniciar el siguiente ciclo de instrucción.

EL CICLO DE EJECUCIÓN

Los ciclos de captación, indirecto y de interrupción son sencillos y predecibles. Cada uno implica una secuencia pequeña y fija de microoperaciones y, en todos los casos, se repiten las mismas microoperaciones para cada ejecución de una instrucción.

Esto no ocurre así en el ciclo de ejecución. En una máquina con *N* códigos de operación diferentes, pueden ocurrir *N* secuencias diferentes de microoperaciones. Consideremos varios ejemplos hipotéticos.

En primer lugar, consideremos una instrucción de suma:

```
ADD  R1, X
```

que suma el contenido de la posición X al registro R1. Puede suceder la siguiente secuencia de microoperaciones:

t_1: MAR ← (IR (dirección))
t_2: MBR ← Memoria
t_3: R1 ← (R1) + (MBR)

En un principio IR contiene la instrucción ADD. En el primer paso, la parte de dirección de IR se carga en MAR. Después se lee la posición de memoria referenciada. Por último, la ALU suma los contenidos de R1 y MBR. De nuevo, este es un ejemplo simplificado. Pueden necesitarse microoperaciones adicionales para extraer la referencia a registro desde IR y tal vez para poner las entradas o salidas de la ALU en algunos registros intermedios.

Consideremos dos ejemplos más complejos. Una instrucción frecuente es «incrementar y saltar si cero» (*Increment and Skip if Zero*):

```
ISZ  X
```

El contenido de la posición X se incrementa en 1. Si el resultado es 0, la siguiente instrucción se salta. Una posible secuencia de microoperaciones es

t_1: MAR ← (IR (dirección))
t_2: MBR ← Memoria
t_3: MBR ← (MBR) + 1
t_4: Memoria ← (MBR)
If ((MBR) = 0) then (PC ← (PC) + *I*)

La nueva característica introducida aquí es la actuación condicional. PC se incrementa si (MBR) = 0. Esta comprobación y actuación puede implementarse como una microoperación. Observe que esta microoperación puede ejecutarse durante la misma unidad de tiempo en la cual el valor actualizado de MBR se almacena en memoria.

Por último, examinemos una instrucción de llamada a subrutina. Como ejemplo, consideremos la instrucción «saltar y guardar la dirección» (*Branch and Save Address*):

```
BSA X
```

La dirección de la instrucción que viene a continuación de la instrucción BSA se guarda en la posición X, y la ejecución continúa en la posición X + *I*. La dirección guardada se usará más adelante en el retorno. Esta es una técnica sencilla para proporcionar llamadas a subrutinas. Son suficientes las siguientes microoperaciones:

t_1: MAR ← (IR (dirección))
MBR ← (PC)
t_2: PC ← (IR (dirección))
Memoria ← (MBR)
t_3: PC ← (PC) + *I*

La dirección que hay en PC al comienzo de la instrucción es la dirección de la siguiente instrucción secuencial. Ésta se guarda en la dirección señalada por IR. Esta última dirección también se incrementa para obtener la dirección de la instrucción correspondiente al siguiente ciclo de instrucción.

EL CICLO DE INSTRUCCIÓN

Hemos visto que cada fase del ciclo de instrucción puede descomponerse en una secuencia de microoperaciones elementales. En nuestro ejemplo, hay una secuencia para cada uno de los ciclos de captación, indirecto y de interrupción, y para el ciclo de ejecución existe una secuencia de microoperaciones para cada código de operación.

Para completar la descripción, tenemos que unir las secuencias de microoperaciones, como se ha hecho en la Figura 16.3. Suponemos que hay un nuevo registro de dos bits llamado *código de ciclo de instrucción* (*instruction cycle code*, ICC). El ICC designa el estado del procesador en términos de en qué parte del ciclo se encuentra este:

00: Captación

01: Indirecto

10: Ejecución

11: Interrupción

Al final de cada uno de los cuatro ciclos, el ICC se actualiza convenientemente. El ciclo indirecto siempre viene seguido del ciclo de ejecución. El ciclo de interrupción siempre es seguido por el

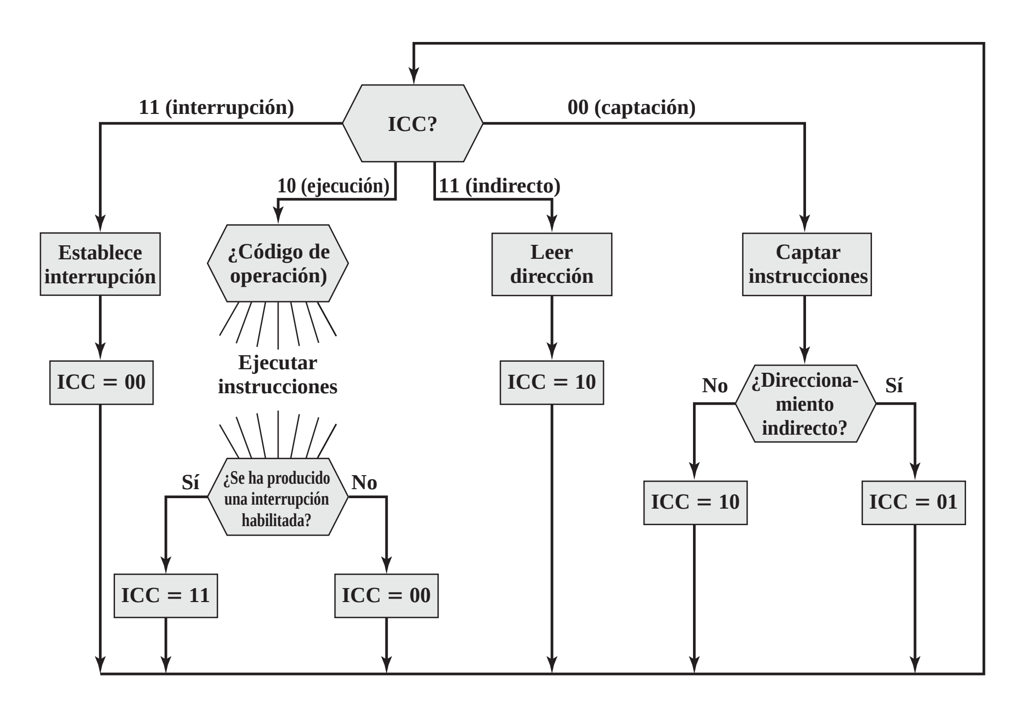


Figura 16.3. Diagrama de flujo del ciclo de instrucción.

ciclo de captación (ver Figura 12.4). En el caso de los ciclos de ejecución y de captación, el siguiente ciclo depende del estado del sistema.

De este modo, el diagrama de flujo del la Figura 16.3 define la secuencia completa de microoperaciones, que dependen solo de la secuencia de instrucciones y del patrón de interrupciones. Naturalmente, este es un ejemplo simplificado. El diagrama de flujo de un procesador real sería más complejo. De todas formas, hemos llegado al punto de nuestra discusión en el que el funcionamiento del procesador se define como la ejecución de una secuencia de microoperaciones. Podemos considerar ahora cómo consigue la unidad de control que ocurra esta secuencia.

16.2. CONTROL DEL PROCESADOR

REQUISITOS FUNCIONALES

Como consecuencia de nuestro análisis de la sección precedente, hemos descompuesto el comportamiento o funcionamiento del procesador en operaciones elementales, llamadas microoperaciones. Reduciendo el funcionamiento del procesador a su nivel más básico, podemos definir exactamente qué es lo que la unidad de control tiene que hacer que ocurra. Así, podemos definir los *requisitos funcionales* de la unidad de control como aquellas funciones que debe llevar a cabo. Una definición de estos requisitos funcionales es la base del diseño e implementación de la unidad de control.

Con la información a mano, el siguiente proceso de tres pasos lleva a la caracterización de la unidad de control:

1. Definir los elementos básicos del procesador.
2. Describir las microoperaciones que ejecuta el procesador.
3. Determinar las funciones que debe realizar la unidad de control para hacer que se ejecuten las microoperaciones.

Ya hemos presentado los pasos 1 y 2. Resumamos los resultados. En primer lugar, los elementos funcionales básicos del procesador son los siguientes:

- ALU
- Registros
- Caminos de datos internos
- Caminos de datos externos
- Unidad de control

Algunas consideraciones deberían convencerle de que esta lista está completa. La ALU es la esencia funcional del computador. Los registros se usan para almacenar datos internos del procesador. Algunos registros contienen información de estado necesaria para gestionar el secuenciamiento de las instrucciones (por ejemplo, la palabra de estado del programa). Otros contienen datos que van hacia, o vienen desde, la ALU, la memoria y los módulos de E/S. Los caminos de datos internos se usan para transferir datos entre los registros y entre estos y la ALU. Los caminos de datos externos unen los registros a la memoria y a los módulos de E/S, a menudo por medio de un bus del sistema. La unidad de control hace que se produzcan operaciones dentro del procesador.

La ejecución de un programa consta de operaciones que involucran estos elementos del procesador. Como hemos visto, estas operaciones consisten en una secuencia de microoperaciones. Tras la revisión de la Sección 16.1, el lector debería observar que todas las microoperaciones se pueden clasificar en una de las siguientes categorías:

- Transferir datos de un registro a otro.
- Transferir datos de un registro a una interfaz externa (por ejemplo, al bus del sistema).
- Transferir datos de una interfaz externa a un registro.
- Realizar una operación aritmética o lógica, usando registros para entrada y salida.

Todas las microoperaciones necesarias para realizar un ciclo de instrucción, incluyendo todas las microoperaciones necesarias para ejecutar cada instrucción del repertorio, están incluidas en una de estas categorías.

Podemos ser ahora algo más explícitos acerca de la forma en que funciona la unidad de control. La unidad de control realiza dos tareas básicas:

- **Secuenciamiento:** la unidad de control hace que el procesador avance a través de una serie de microoperaciones en la secuencia oportuna, basada en el programa que se está ejecutando.
- **Ejecución:** la unidad de control hace que se ejecute cada microoperación.

Lo que precede es una descripción funcional de lo que hace la unidad de control. La clave de cómo funciona la unidad de control es la utilización de señales de control.

SEÑALES DE CONTROL

Hemos definido los elementos que componen el procesador (ALU, registros, caminos de datos) y las microoperaciones que se llevan a cabo. Para que la unidad de control realice su función, debe tener entradas que le permitan determinar el estado del sistema y salidas que le permitan controlar el comportamiento del mismo. Estas son las especificaciones externas de la unidad de control. Internamente, la unidad de control ha de tener la lógica necesaria para realizar sus funciones de secuenciamiento y ejecución. Aplazamos el estudio del funcionamiento interno de la unidad de control hasta la Sección 16.3 y el Capítulo 17. El resto de esta sección se ocupa de la interacción entre la unidad de control y otros elementos del procesador.

La Figura 16.4 es un modelo general de la unidad de control, que muestra todas sus entradas y salidas. Las entradas son las siguientes:

- **Reloj:** es el encargado de «mantener la hora exacta». La unidad de control hace que se ejecute una microoperación (o un conjunto de microoperaciones simultáneas) en cada pulso de reloj. Este a menudo es referenciado como tiempo de ciclo del procesador, o período de reloj.
- **Registro de instrucción:** el código de operación de la instrucción en curso se usa para determinar qué microoperaciones hay que realizar durante el ciclo de ejecución.
- **Indicadores:** los necesita la unidad de control para determinar el estado del procesador y el resultado de anteriores operaciones de la ALU. Por ejemplo, para la instrucción incrementar y saltar si cero (ISZ), la unidad de control incrementará PC si el indicador de cero está a uno.
- **Señales de control del bus de control:** la parte de control del bus del sistema suministra señales a la unidad de control, tales como señales de interrupción y de reconocimiento.

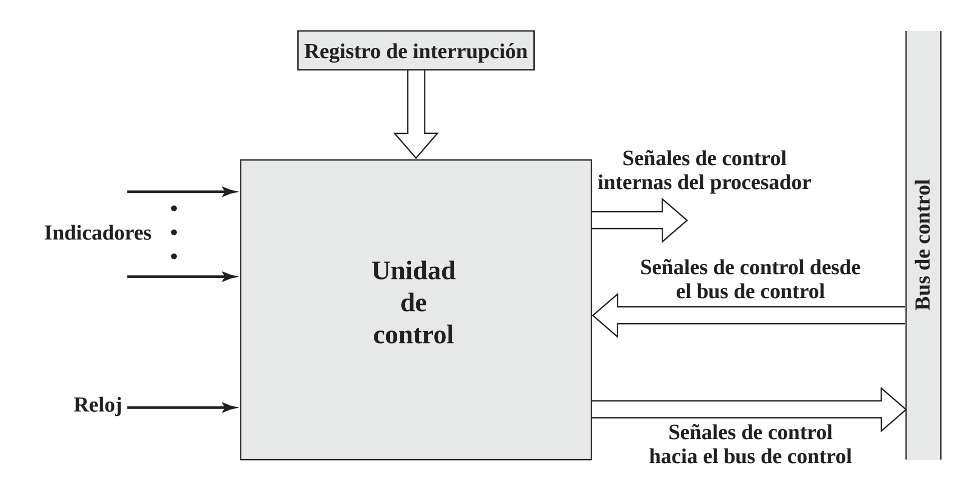


Figura 16.4. Diagrama de bloques de la unidad de control.

Las salidas son las siguientes:

- **Señales de control internas al procesador:** son de dos tipos: las que hacen que los datos se transfieran de un registro a otro, y las que activan funciones específicas de la ALU.
- **Señales de control hacia el bus de control***:* también las hay de dos tipos: señales de control de la memoria, y señales de control de los módulos de E/S.

El nuevo elemento introducido en esta figura es el de señal de control. Se usan tres tipos de señales de control: las que activan una función de la ALU, las que activan un camino de datos, y las que son señales del bus del sistema externo u otra interfaz externa. Todas estas señales acaban aplicándose directamente como entradas binarias a puertas lógicas individuales.

Consideremos nuevamente el ciclo de captación para entender cómo mantiene el control la unidad de control. La unidad de control se mantiene al tanto de dónde está dentro del ciclo de instrucción. En un punto determinado, sabe que inmediatamente después se va a realizar un ciclo de captación. El primer paso es transferir el contenido de PC a MAR. La unidad de control hace esto activando la señal de control que abre las puertas entre los bits de PC y los bits de MAR. El siguiente paso es leer una palabra desde memoria a MBR e incrementar PC. La unidad de control hace esto enviando las siguientes señales de control simultáneamente:

- Una señal de control que abre las puertas que permiten que el contenido de MAR aparezca en el bus de direcciones.
- Una señal de control de lectura de memoria, en el bus de control.
- Una señal de control que abre las puertas que permiten almacenar el contenido del bus de datos en MBR.
- Señales de control de la lógica que suma *I* al contenido de PC y almacena el resultado de nuevo en PC.

Después de esto, la unidad de control envía una señal de control que abre las puertas adecuadas entre MBR e IR.

Esto completa el ciclo de captación exceptuando un detalle: la unidad de control debe decidir si ejecuta a continuación un ciclo indirecto o un ciclo de ejecución. Para decidir esto, examina IR viendo si se hace una referencia indirecta a memoria.

Los ciclos indirecto y de interrupción funcionan de un modo parecido. En el caso del ciclo de ejecución, la unidad de control comienza examinando el código de operación y, en función de él, decide qué secuencia de microoperaciones deben realizarse para ejecutar el ciclo.

UN EJEMPLO DE SEÑALES DE CONTROL

Para ilustrar el funcionamiento de la unidad de control, examinemos un ejemplo sencillo. La Figura 16.5 ilustra el ejemplo. Se trata de un procesador sencillo con un único acumulador. Se indican los caminos de datos entre los distintos elementos. Los caminos de control de las señales que proceden de la unidad de control no se muestran, pero las terminaciones de las señales de control están

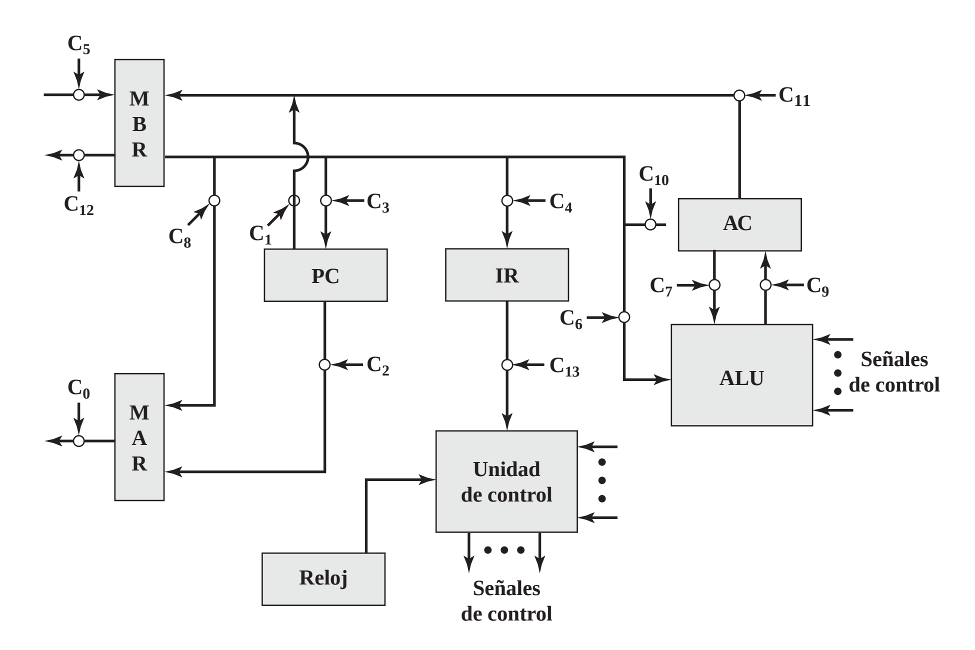


Figura 16.5. Caminos de datos y señales de control.

designadas como C_i y se indican mediante un círculo. La unidad de control recibe entradas del reloj, del registro de instrucción, y de los indicadores. En cada ciclo de reloj, la unidad de control lee todas sus entradas y emite un conjunto de señales de control. Las señales de control van hacia tres destinos distintos:

- **Caminos de datos:** la unidad de control dirige el flujo interno de datos. Por ejemplo, en la captación de instrucción, el contenido del registro intermedio de memoria se transfiere al registro de instrucción. Por cada camino a controlar hay una puerta (indicada mediante un círculo en la figura). Una señal de control de la unidad de control abre temporalmente la puerta para dejar pasar los datos.
- **ALU:** la unidad de control gobierna el funcionamiento de la ALU mediante un conjunto de señales de control. Estas señales activan diversos dispositivos y puertas dentro de la ALU.
- **Bus del sistema:** la unidad de control envía señales de control a las líneas de control del bus del sistema (por ejemplo, lectura de la memoria).

La unidad de control debe conocer en todo momento dónde está dentro del ciclo de instrucción. Usando ese conocimiento, y leyendo todas sus entradas, la unidad de control emite una serie de señales de control que hacen que se efectúen las microoperaciones. La unidad de control usa los pulsos de reloj para temporizar la secuencia de eventos, dejando tiempo entre eventos para que los niveles de las señales se estabilicen. La Tabla 16.1 indica las señales de control necesarias para realizar algunas de las secuencias de microoperaciones descritas con anterioridad. Por simplicidad, no se han mostrado los caminos de datos y de control necesarios para incrementar PC y para cargar direcciones fijas en PC y MAR.

El carácter mínimo de la unidad de control merece ser considerado. La unidad de control es el motor que mueve todo el computador. Lo hace basándose solo en el conocimiento de las instrucciones

Tabla 16.1. Microoperaciones y señales de control.

Microoperaciones	Temporización	Señales de control activas
Captación:	t_1: MAR ← (PC)	C_2
	t_2: MBR ← Memoria PC ← (PC) + *I*	C_5, C_R
	t_3: IR ← (MBR)	C_4
Indirecto:	t_1: MAR ← (IR (dirección))	C8
	t_2: MBR ← Memoria	C_5, C_R
	t_3: IR (dirección) ← (MBR (dirección))	C_4
Interrupción:	t_1: MBR ← (PC)	C_1
	t_2: MAR ← Dirección de salvaguardia PC ← Dirección de rutina	
	t_3: Memoria ← (MBR)	C_{12}, C_W

C_R = Señal de control de lectura (*read*) hacia el bus del sistema.
C_W = Señal de control de escritura (*write*) hacia el bus del sistema.

que tiene que ejecutar y de la naturaleza de los resultados de las operaciones aritméticas y lógicas (por ejemplo, resultado positivo, desbordamiento, etc.). Nunca llega a ver los datos que se procesan o los resultados reales producidos. Y controla todo con unas pocas señales de control que van a ciertos puntos dentro del procesador y unas pocas señales que van hacia el bus del sistema.

ORGANIZACIÓN INTERNA DEL PROCESADOR

La Figura 16.5 indica el uso de diversos caminos de datos. La complejidad de este tipo de organización debería estar clara. Es más normal usar algún tipo de configuración de bus interno, como se sugirió en la Figura 12.2.

Usando un bus interno al procesador, la Figura 16.5 puede disponerse del modo que muestra la Figura 16.6. Un único bus interno conecta la ALU y todos los registros del procesador. Hay puertas y señales de control encargadas de realizar transferencias de datos entre el bus y cada registro. Otras señales de control dirigen las transferencias de datos hacia y desde el bus (externo) del sistema y el funcionamiento de la ALU.

Se han añadido dos nuevos registros, rotulados como Y y Z, a la organización. Son necesarios para el correcto funcionamiento de la ALU. Cuando se realiza una operación que incluye dos operandos, uno se puede obtener desde el bus interno, pero el otro ha de obtenerse de otra fuente. El AC podría usarse para este propósito, pero ello limita la flexibilidad del sistema y no funcionaría en un procesador con múltiples registros de uso general. El registro Y proporciona un almacenamiento temporal de la otra entrada. La ALU es un circuito combinacional (*ver* Apéndice A) sin almacenamiento interno. De este modo, cuando las señales de control activan una función de la ALU, la entrada de esta se transforma en una salida. Debido a ello, la salida de la ALU no se puede conectar directamente al bus, ya

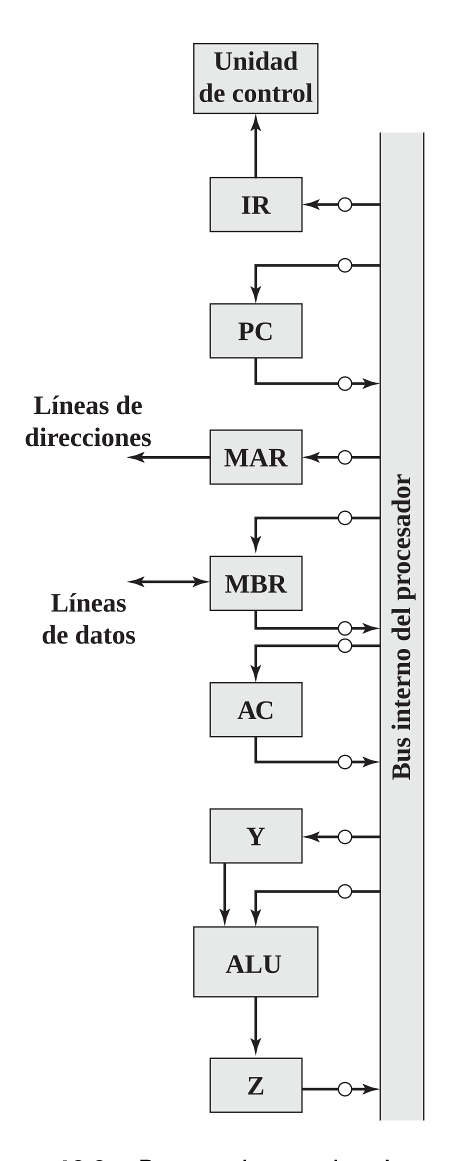


Figura 16.6. Procesador con bus interno.

que realimentaría la entrada. El registro Z proporciona el almacenamiento temporal de salida. Con esta configuración, una operación de suma de un valor de la memoria al AC tendría los siguientes pasos:

t_1: MAR ← (IR (dirección))
t_2: MBR ← Memoria
t_3: Y ← (MBR)
t_4: Z ← (AC) + (Y)
t_3: R1 ← (R1) + (MBR)

Son posibles otras organizaciones, pero, en general, se usa algún tipo de bus interno o conjunto de buses internos. El uso de caminos de datos comunes simplifica el trazado de las interconexiones y

el control del procesador. Otra razón práctica para usar un bus interno es ahorrar espacio. El espacio ocupado por las conexiones entre registros tiene que minimizarse especialmente en los microprocesadores, que sólo pueden ocupar un trozo cuadrado de silicio de un cuarto de pulgada.

EL INTEL 8085

Para ilustrar algunos de los conceptos introducidos hasta aquí en este capítulo, consideremos el Intel 8085. Su organización se muestra en la Figura 16.7. Hay varios componentes clave que pueden no explicarse por sí mismos:

- ***Latch* (cerrojo) incrementador/decrementador de direcciones:** lógica que puede sumar 1 o restar 1 al contenido del puntero de pila o al contador de programa. Ahorra tiempo ya que evita usar la ALU para este fin.
- **Control de interrupciones:** este módulo maneja múltiples niveles de señales de interrupción.

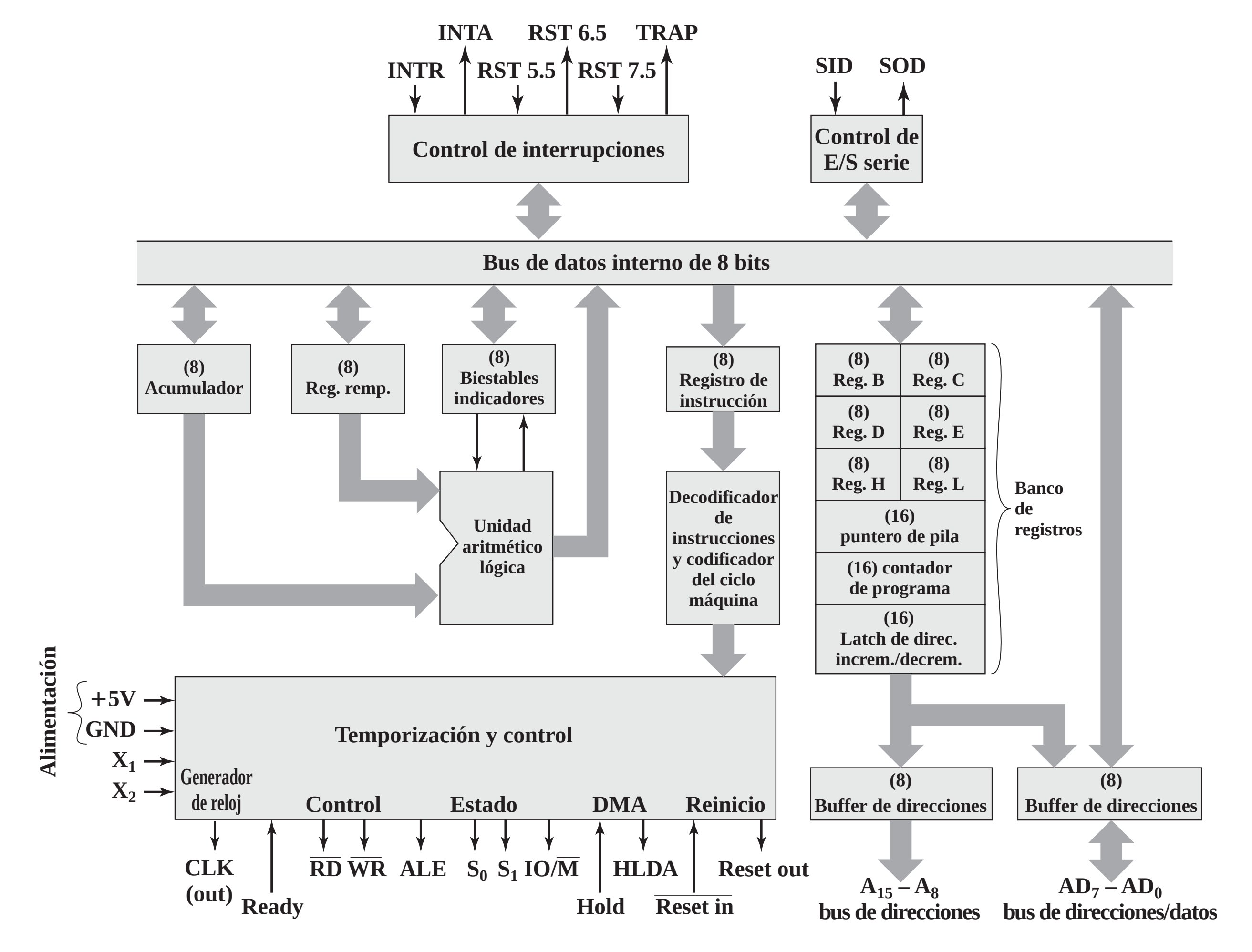


Figura 16.7. Diagrama de bloques del procesador Intel 8085.

- **Control de E/S serie:** este módulo se conecta a dispositivos capaces de transferir 1 bit cada vez.

La Tabla 16.2 describe las señales externas que entran y salen del 8085. Estas se unen al bus externo del sistema y son la interfaz entre el procesador 8085 y el resto del sistema (Figura 16.8).

Tabla 16.2. Señales externas del Intel 8085.

Señales de datos y direcciones

Direcciones altas (A_{15} – A_8)
Los ocho bits más significativos de una dirección de 16 bits.

Direcciones/datos (AD7 – AD0)
Los ocho bits menos significativos de una dirección de 16 bits o bien ocho bits de datos. Esta multiplexación ahorra pines.

Dato de entrada serie (*Serial Input Data,* SID)
Una entrada de un único bit para adaptarse a dispositivos que transmitan de forma serie (un bit cada vez).

Dato de salida serie (*Serial Output Data,* SOD)
Una salida de un único bit para adaptarse a dispositivos que reciben de forma serie.

Señales de temporización y control

CLK (*Out*)
El reloj del sistema. Cada ciclo representa un estado T. La señal CLK va hacia circuitos periféricos y sincroniza su temporización.

X_1, X_2
Estas señales provienen de un cristal externo u otro dispositivo para controlar el generador de reloj interno.

Habilitación del latch de direcciones (*Address Latch Enable,* ALE)
Tiene lugar durante el primer estado de un ciclo máquina y hace que los circuitos periféricos almacenen las líneas de dirección. Esto permite a un módulo (por ejemplo, memoria, E/S) reconocer que está siendo direccionado.

Estado (S_0, S_1)
Señales de control usadas para indicar si está teniendo lugar una operación de lectura o escritura.

IO/$\overline{M}$
Usada para controlar operaciones de lectura o escritura de los módulos de E/S o de memoria.

Control de lectura ($\overline{RD}$)
Indica que el módulo de memoria o de E/S seleccionado va a leerse y que el bus de datos está disponible para una transferencia de datos.

Control de escritura ($\overline{WR}$)
Indica que el dato que hay en el bus de datos va a escribirse en la posición de memoria o E/S seleccionada.

Señales originadas en memoria y E/S

Hold
Pide al procesador que abandone el control y el uso del bus del sistema externo. El procesador completará la ejecución de la instrucción que hay en el momento presente en el IR y entrará en un estado de desconexión, durante el cual no pone señales en los buses de control, de direcciones y de datos. Durante el estado de desconexión, el bus puede usarse para operaciones de DMA.

Reconocimiento de Hold (*Hold Acknowledge,* HLDA)
Esta señal de salida de la unidad de control reconoce la señal HOLD e indica que el bus queda disponible.

Ready
Usado para sincronizar el procesador con una memoria más lenta o con dispositivos de E/S. Cuando un dispositivo seleccionado mantiene Ready a uno, el procesador puede continuar con una operación de entrada (DBIN) o salida (WR). En caso contrario, el procesador entra en un estado de espera hasta que el dispositivo esté listo.

(Continúa)

Tabla 16.2. Señales externas del Intel 8085 *(continuación)*.

Señales relacionadas con interrupciones

Trap.

Interrupciones de reanudación (RST 7.5, 6.5, 5.5)

Petición de interrupción (INTR)

Un dispositivo externo usa estas cinco líneas para interrumpir al procesador. Este no aceptará una petición si se encuentra en el estado de desconexión o si la interrupción está inhabilitada. Una interrupción se acepta solo al final de una instrucción. Estas interrupciones se indican en orden de prioridad descendente.

Reconocimiento de interrupción

Reconoce una interrupción.

Reinicio del procesador

$\overline{\text{Reset In}}$

Provoca la puesta a cero del contenido de PC. El procesador reanuda la ejecución en la posición cero.

Reset Out

Reconoce que el procesador ha sido reiniciado. Esta señal puede usarse para reiniciar el resto del sistema.

Alimentación y tierra

V_{CC}

Alimentación a +5 voltios.

V_{SS}

Tierra.

La unidad de control se identifica como los dos componentes rotulados (1) decodificador de instrucciones y codificación del ciclo máquina y (2) temporización y control. Se aplaza la discusión del primer componente hasta la siguiente sección. La parte fundamental de la unidad de control es el módulo de temporización y control. Este módulo incluye un reloj y acepta como entradas la instrucción en curso y algunas señales de control externas. Su salida consiste en señales de control hacia los otros componentes del procesador más señales de control hacia el bus externo del sistema.

La temporización de las operaciones del procesador está sincronizada por el reloj y está controlada por la unidad de control por medio de señales de control. Cada ciclo de instrucción se divide en *ciclos máquina*, de uno a cinco; cada ciclo máquina se divide a su vez en *estados*, de tres a cinco. Cada estado dura un ciclo de reloj. Durante un estado, el procesador ejecuta una o un conjunto de microoperaciones simultáneas determinadas por las señales de control.

El número de ciclos máquina es fijo para cada instrucción pero varía de una instrucción a otra. Los ciclos máquina se definen como equivalentes a los accesos al bus. De este modo, el número de ciclos máquina de una instrucción depende del número de veces que el procesador debe comunicarse con los dispositivos externos. Por ejemplo, si una instrucción se compone de dos partes de ocho bits, se necesitan dos ciclos máquina para captar dicha instrucción. Si la instrucción implica una operación de un byte con memoria o E/S, será necesario un tercer ciclo máquina para su ejecución.

La Figura 16.9 ofrece un ejemplo de temporización del 8085 que muestra el valor de las señales de control externas. Naturalmente, al mismo tiempo, la unidad de control está generando señales de control internas para controlar transferencias de datos en el interior del procesador. El diagrama muestra el ciclo de instrucción de la instrucción OUT. Se necesitan tres ciclos máquina (M_1, M_2, M_3).

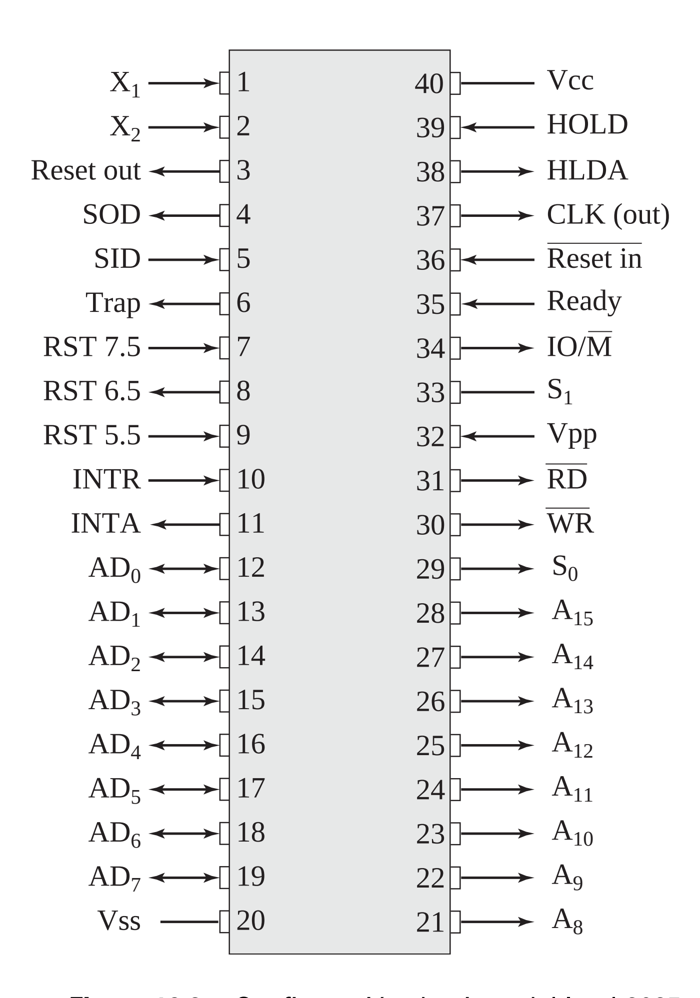


Figura 16.8. Configuración de pines del Intel 8085.

Durante el primero se capta la instrucción OUT. El segundo ciclo máquina capta la segunda mitad de la instrucción, que contiene el número del dispositivo de E/S seleccionado para salida. Durante el tercer ciclo, el contenido del AC se escribe a través del bus de datos en el dispositivo seleccionado.

El comienzo de cada ciclo máquina viene determinado por el pulso de habilitación del *latch* de direcciones (*Address Latch Enable*, ALE) emitido por la unidad de control. Durante el estado de temporización T_1 del ciclo máquina M_1, la unidad de control ajusta la señal $IO/\overline{M}$ para indicar una operación con memoria. Además, la unidad de control hace que el contenido de PC se sitúe en el bus de direcciones (A_{15} a A_8) y el bus de direcciones/datos (AD_7 a AD_0). En el flanco de bajada del pulso ALE, los otros módulos conectados al bus almacenan la dirección.

Durante el estado de temporización T_2, el módulo de memoria direccionado pone el contenido de la posición de memoria en el bus de direcciones/datos. La unidad de control activa la señal de control de lectura ($\overline{RD}$) para indicar una lectura, pero espera hasta T_3 para captar los datos del bus. Esto deja tiempo al módulo de memoria para poner los datos en el bus y para que se estabilicen los niveles de las señales. El estado final, T_4, es un estado de *bus desocupado* durante el cual el procesador decodifica la instrucción. Los restantes ciclos máquina actúan de forma parecida.

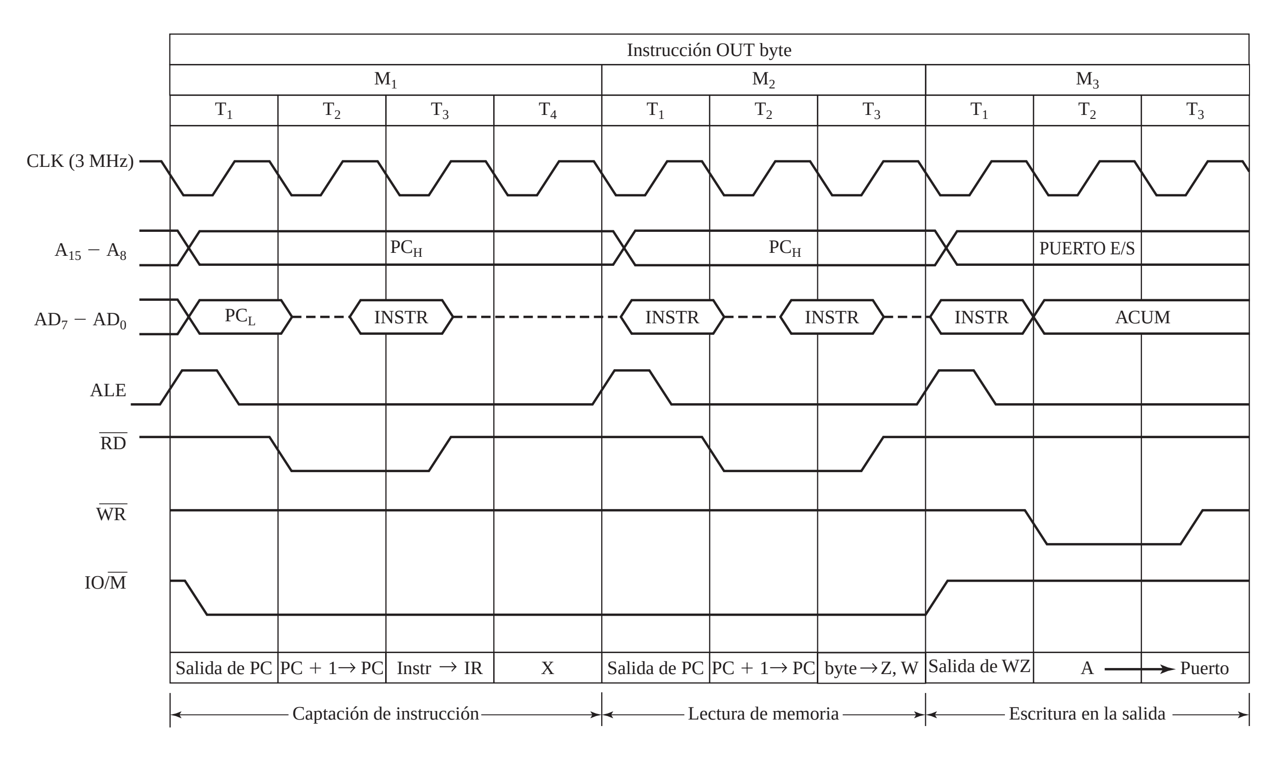


Figura 16.9. Diagrama de tiempos de la instrucción OUT del Intel 8085.

16.3. IMPLEMENTACIÓN CABLEADA

Hemos estudiado la unidad de control en lo referente a entradas, salidas y funciones. Ahora es el momento de volver al tema de la implementación de la unidad de control. Se ha usado una gran variedad de técnicas. La mayoría de ellas se pueden clasificar en dos categorías:

- Implementación cableada
- Implementación microprogramada

En una implementación cableada, la unidad de control es esencialmente un circuito combinacional. Sus señales lógicas de entrada se transforman en un conjunto de señales lógicas de salida, que son las señales de control. Este enfoque se examina en esta sección. La implementación microprogramada es el tema del que trata el Capítulo 17.

ENTRADAS DE LA UNIDAD DE CONTROL

La Figura 16.4 representa la unidad de control como la hemos estudiado hasta aquí. Las entradas principales son el registro de instrucción, el reloj, los indicadores y las señales de control del bus. En el caso de los indicadores y de las señales de control del bus, cada bit individual tiene normalmente

un significado determinado (por ejemplo, desbordamiento). Las otras dos entradas, sin embargo, no son útiles a la unidad de control tal como entran.

Consideremos en primer lugar el registro de instrucción. La unidad de control hace uso del código de operación y realiza acciones diferentes (emite una combinación diferente de señales de control) para cada instrucción. Para simplificar la lógica de la unidad de control, debería existir una entrada lógica única para cada código de operación. Esta función la puede realizar un *decodificador*, que toma una entrada codificada y produce una salida única. En general, un decodificador tendrá n entradas binarias y 2^n salidas binarias. Cada uno de los 2^n patrones de entrada distintos activará una única salida. La Tabla 16.3 es un ejemplo. El decodificador de una unidad de control normalmente tendrá que ser más complejo que el de la tabla, para representar códigos de operación de longitud variable. En el Apéndice A se presenta un ejemplo de la lógica digital usada para realizar un decodificador.

El reloj de la unidad de control emite una secuencia repetitiva de pulsos. Esto es útil para delimitar la duración de las microoperaciones. Esencialmente, el periodo de los pulsos de reloj ha de ser suficientemente largo para permitir la propagación de las señales a lo largo de los caminos de datos y a través de la circuitería del procesador. Sin embargo, como hemos visto, la unidad de control emite señales de control diferentes en unidades de tiempo diferentes dentro de un único ciclo de instrucción. Por tanto, podríamos tener un contador como entrada a la unidad de control, con una señal de control diferente para T_1, T_2, etc. Al final de un ciclo de instrucción, la unidad de control deberá realimentar el contador para reiniciarlo a T_1.

Con estos dos refinamientos, la unidad de control se puede representar como en la Figura 16.10.

Tabla 16.3. Un decodificador con cuatro entradas y dieciséis salidas.

I1	I2	I3	I4	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11	O12	O13	O14	O15	O16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

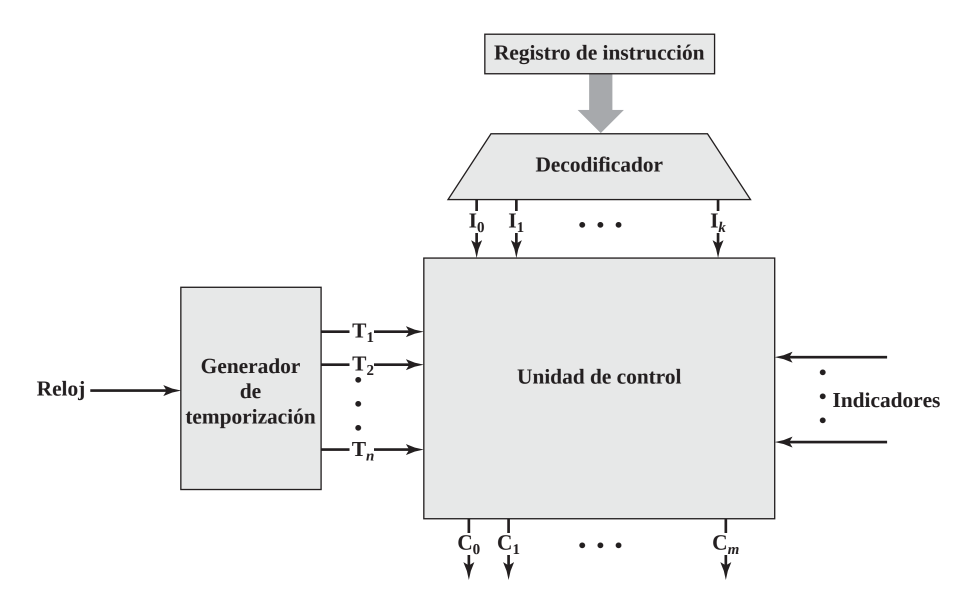


Figura 16.10. Unidad de control con entradas decodificadas.

LÓGICA DE LA UNIDAD DE CONTROL

Para definir la implementación cableada de una unidad de control, todo lo que queda es estudiar su lógica interna, que produce señales de control de salida a partir de las señales de entrada.

Básicamente, lo que tenemos que hacer es, para cada señal de control, obtener su expresión booleana como una función de las entradas. Esto se explica mejor con un ejemplo. Consideremos otra vez nuestro ejemplo sencillo ilustrado en la Figura 16.5. Vimos en la Tabla 16.1 las secuencias de microoperaciones y de señales de control necesarias para controlar tres de las cuatro fases del ciclo de instrucción.

Consideremos una única señal de control, C_5. Esta señal hace que se lean datos del bus de datos externo en MBR. Podemos ver que se usa dos veces en la Tabla 16.1. Definamos dos nuevas señales de control, P y Q, que tengan la siguiente interpretación:

PQ = 00	Ciclo de captación
PQ = 01	Ciclo indirecto
PQ = 10	Ciclo de ejecución
PQ = 11	Ciclo de interrupción

La siguiente expresión booleana define a C_5:

$$C_5 = \overline{P} \cdot \overline{Q} \cdot T_2 + \overline{P} \cdot Q \cdot T_2$$

Es decir, la señal de control C_5 se pondrá a uno durante la segunda unidad de tiempo de los ciclos de captación e indirecto.

Esta expresión no está completa. C_5 se necesita también durante el ciclo de ejecución. Para nuestro sencillo ejemplo, supongamos que hay solo tres instrucciones que leen de la memoria: LDA, ADD, y AND. Ahora podemos definir C_5 como

$$C_5 = \overline{P} \cdot \overline{Q} \cdot T_2 + \overline{P} \cdot Q \cdot T_2 + P \cdot \overline{Q} \cdot (LDA + ADD + AND) \cdot T_2$$

Este mismo proceso podría repetirse para cada señal de control generada por el procesador. El resultado sería un conjunto de ecuaciones booleanas que definiría el comportamiento de la unidad de control y por tanto del procesador.

Juntando todo, la unidad de control debe controlar el estado del ciclo de instrucción. Como se mencionó, al final de cada subciclo (captación, indirecto, ejecución, interrupción), la unidad de control emite una señal que hace que el generador de temporización se reinicie y emita T_1. Además, la unidad de control ha de establecer los valores adecuados de P y Q para definir el siguiente subciclo a ejecutar.

El lector debe comprender que en un complejo procesador moderno, el número de ecuaciones booleanas necesarias para definir la unidad de control es muy grande. La tarea de implementar un circuito combinacional que satisfaga todas esas ecuaciones llega a ser sumamente difícil. El resultado es que, por regla general, se usa una aproximación mucho más sencilla, conocida como *microprogramación*. De este tema se ocupa el próximo capítulo.

16.4. LECTURAS RECOMENDADAS

Varios libros de texto tratan los principios básicos del funcionamiento de la unidad de control, entre los que se incluyen [FARH04] y [MANO04].

FARH04 Farhat, H.: *Digital Design and Computer Organization.* Boca Raton, FL. CRC Press, 2004.

MANO04 Mano, M.: *Logic and Computer Design Fundamentals.* Upper Saddle River, NJ. Prentice Hall, 2004.

16.5. PALABRAS CLAVE, PREGUNTAS DE REPASO Y PROBLEMAS

PALABRAS CLAVE

bus de control	implementación cableada	señal de control
camino de control	microoperaciones	unidad de control

PREGUNTAS DE REPASO

16.1. Explique la diferencia entre la secuencia escrita y la secuencia de tiempo de una instrucción.

16.2. ¿Cuál es la relación entre instrucciones y microoperaciones?

16.3. ¿Cuál es la función general de la unidad de control de un procesador?

16.4. Bosqueje un proceso de tres pasos que conduzca a la caracterización de la unidad de control.

16.5. ¿Qué tareas básicas realiza una unidad de control?

16.6. Escriba una lista típica de entradas y salidas de una unidad de control.

16.7. Indique tres tipos de señales de control.

16.8. Explique brevemente qué se entiende por una implementación cableada de una unidad de control.

PROBLEMAS

16.1. Dispone de una ALU que puede sumar sus dos registros de entrada, y puede hacer la negación lógica de los bits de cada registro de entrada, pero no puede restar. Los números se van a almacenar en complemento a dos. Enumere las microoperaciones que debe realizar la ALU para hacer una resta.

16.2. Muestre las microoperaciones y señales de control, tal como lo hace la Tabla 16.1 para el procesador de la Figura 16.5, para las siguientes instrucciones:

- Cargar el acumulador
- Almacenar el acumulador
- Sumar al acumulador
- AND con el acumulador
- Saltar
- Saltar si AC = 0
- Complementar el acumulador

16.3. Suponga que los retardos de propagación a lo largo del bus y a través de la ALU de la Figura 16.6 son 20 y 100 ns, respectivamente. El tiempo necesario para la carga de datos en un registro desde el bus es 10 ns. ¿Cuál es el tiempo que debe asignarse a

(a) transferir datos de un registro a otro?
(b) incrementar el contador de programa?

16.4. Escriba la secuencia de microoperaciones necesaria en la estructura de bus de la Figura 16.6 para sumar un número al AC si el número es

(a) un operando inmediato
(b) un operando direccionado directamente
(c) un operando direccionado indirectamente

16.5. Una pila está organizada como se muestra en la Figura 10.14. Muestre la secuencia de microoperaciones para

(a) desapilar
(b) apilar

CAPÍTULO 17

Control microprogramado

PUNTOS CLAVE

- Una alternativa a la unidad de control cableada es la unidad de control microprogramada, en la cual la lógica de la unidad de control se especifica mediante un microprograma. Un microprograma consiste en una secuencia de instrucciones en un lenguaje de microprogramación. Se trata de instrucciones muy elementales que especifican microoperaciones.
- Una unidad de control microprogramada es un circuito lógico relativamente sencillo que es capaz de (1) realizar el secuenciamiento de las microinstrucciones y (2) generar las señales de control para ejecutar cada microinstrucción.
- Como en una unidad de control cableada, las señales de control generadas por una microinstrucción se usan para producir transferencias entre registros y operaciones de la ALU.

El término *microprograma* lo acuñó M. V. Wilkes a principios de los años cincuenta [WILK51]. Wilkes propuso una aproximación al diseño de la unidad de control que era ordenada y sistemática, y evitaba la complejidad de la implementación cableada. La idea despertó la curiosidad de muchos investigadores pero se mostraba irrealizable porque necesitaba una memoria de control que fuera rápida y relativamente barata.

El número de febrero de 1964 de *Datamation* revisaba el estado del arte de la microprogramación. No había ningún sistema microprogramado de uso generalizado en aquella época, y uno de los artículos [HILL64] resumía la opinión popular del momento de que el futuro de la microprogramación «es un tanto turbio. Ninguno de los grandes fabricantes ha mostrado interés en la técnica, a pesar de que probablemente todos la hayan analizado».

La situación cambió completamente en muy pocos meses. El System/360 de IBM se anunció en abril, y todos excepto los modelos más grandes eran microprogramados. Aunque la serie 360 precedió a la disponibilidad de ROM de semiconductores, las ventajas de la microprogramación fueron lo bastante convincentes para IBM como para introducir este cambio. La microprogramación se convirtió en una técnica popular para implementar la unidad de control de los procesadores CISC. En los últimos años, la microprogramación se está usando menos, pero sigue siendo una herramienta disponible para los diseñadores de computadores. Por ejemplo, tal y como hemos visto, en el Pentium 4, las instrucciones máquina se convierten a un formato tipo RISC y la mayoría de estas nuevas instrucciones se ejecutan sin el uso de microprogramación. No obstante, algunas de las instrucciones se ejecutan usando microprogramación.

17.1. CONCEPTOS BÁSICOS

MICROINSTRUCCIONES

La unidad de control parece un dispositivo bastante sencillo. Sin embargo, implementar una unidad de control como una interconexión de elementos lógicos básicos no es una tarea fácil. El diseño debe

incluir lógica para realizar el secuenciamiento de las microoperaciones, para ejecutar las microoperaciones, para interpretar los códigos de operación, y para tomar decisiones basadas en los indicadores de la ALU. Todo este hardware es difícil de diseñar y de verificar. Además, el diseño es relativamente inflexible. Por ejemplo, es difícil cambiar el diseño si se desea añadir una nueva instrucción máquina.

Una alternativa, que se ha usado en muchos procesadores CISC, es realizar una unidad de control microprogramada.

Consideremos de nuevo la Tabla 16.1. Además de indicar las señales de control, cada microoperación está descrita en notación simbólica. Esta notación tiene toda la apariencia de un lenguaje de programación. De hecho es un lenguaje, conocido como **lenguaje de microprogramación**. Cada línea describe un conjunto de microoperaciones que suceden a la vez y que se conoce como **microinstrucción**. Una secuencia de instrucciones se conoce como **microprograma** o *firmware*. Este último término refleja el hecho de que un microprograma está a mitad de camino entre hardware y software. Es más fácil diseñar en firmware que en hardware, pero es más difícil escribir un programa firmware que un programa software.

¿Cómo se puede usar el concepto de microprogramación para implementar una unidad de control? Consideremos que para cada microoperación, todo lo que la unidad de control puede hacer es generar un conjunto de señales de control. Por consiguiente, para cualquier microoperación, cada línea de control procedente de la unidad de control estará activa o inactiva. Esta condición puede, naturalmente, representarse con un dígito binario para cada línea de control. De este modo, podríamos construir una *palabra de control* en la que cada bit representara una línea de control. Entonces, cada microoperación se representaría mediante un patrón diferente de unos y ceros en la palabra de control.

Supongamos que se emplea una secuencia de palabras de control para representar la secuencia de microoperaciones ejecutadas por la unidad de control. A continuación, hemos de admitir que la secuencia de microoperaciones no es fija. Algunas veces tenemos un ciclo indirecto; otras no. Por tanto, coloquemos nuestras palabras de control en una memoria, cada palabra en una dirección única. Añadamos ahora un campo de dirección a cada palabra de control, indicando la posición de la siguiente palabra de control a ejecutar si una determinada condición es cierta (por ejemplo, que el bit de direccionamiento indirecto de una instrucción que referencia la memoria sea uno). Y añadamos además algunos bits para especificar la condición.

El resultado se conoce como **microinstrucción horizontal**; se muestra un ejemplo en la Figura 17.1a. El formato de la microinstrucción o palabra de control es el siguiente. Hay un bit para cada línea de control interna al procesador y un bit para cada línea de control del bus del sistema. Hay un campo de condición que indica la condición bajo la cual debe producirse un salto, y un campo con la dirección de la microinstrucción a ejecutar cuando el salto se produzca. Esta microinstrucción se interpreta como sigue:

1. Para ejecutar la microinstrucción, se activan todas las líneas de control cuyos bits estén a 1; y se dejan inactivas todas las líneas de control indicadas con un bit a 0. Las señales de control resultantes harán que se ejecuten una o más microoperaciones.

2. Si la condición indicada por los bits de condición es falsa, se ejecuta la siguiente microinstrucción secuencial.

3. Si la condición indicada por los bits de condición es cierta, la siguiente microinstrucción a ejecutar se indica en el campo de dirección.

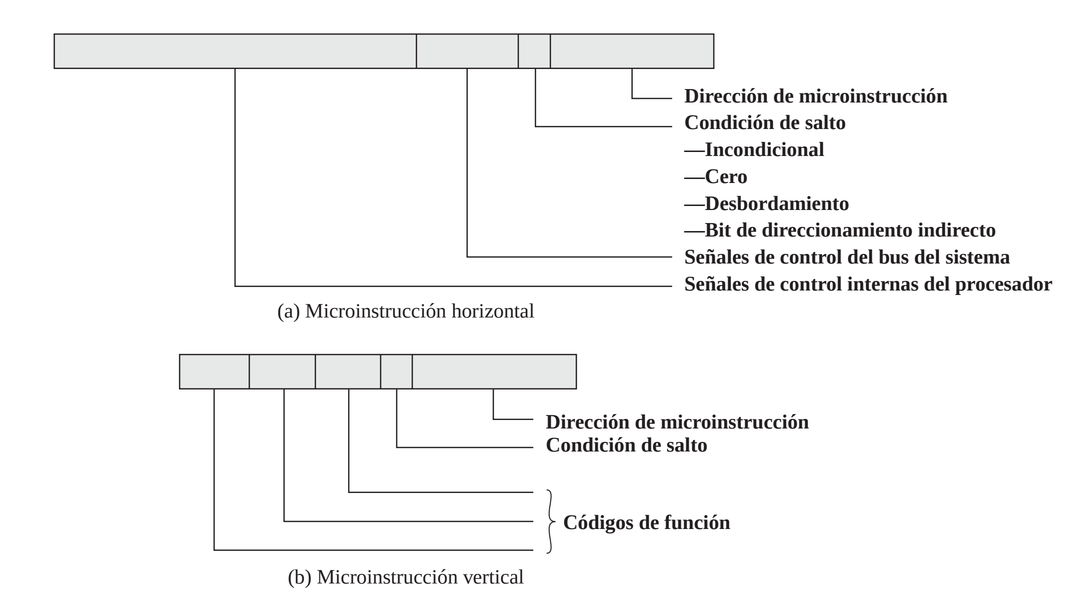


Figura 17.1. Formatos de microinstrucción típicos.

La Figura 17.2 muestra cómo se pueden organizar estas palabras de control o microinstrucciones en una **memoria de control**. Las microinstrucciones de cada rutina se ejecutarán secuencialmente. Cada rutina termina con una instrucción de bifurcación o salto indicando a dónde ir a continuación. Hay una sección especial del ciclo de ejecución cuyo único objetivo es indicar cuál de las rutinas de las instrucciones máquina (AND, ADD, etc.) se va a ejecutar a continuación, en función del código de operación actual.

La memoria de control de la Figura 17.2 es una descripción concisa de todo lo que hace la unidad de control. Define la secuencia de microoperaciones a realizar en cada ciclo (captación, indirecto, ejecución, interrupción), y especifica el secuenciamiento de estos ciclos. Si solo fuera eso, esta notación sería un recurso útil para documentar el funcionamiento de una unidad de control para un computador particular. Pero es más que eso. Es también una forma de implementar la unidad de control.

UNIDAD DE CONTROL MICROPROGRAMADA

La memoria de control de la Figura 17.2 contiene un programa que describe el funcionamiento de la unidad de control. Resulta que podríamos implementar la unidad de control sencillamente ejecutando ese programa.

La Figura 17.3 muestra los elementos más importantes de esta implementación. El conjunto de microinstrucciones se almacena en la *memoria de control*. El *registro de dirección de control* contiene la dirección de la siguiente microinstrucción a leer. Cuando se lee una microinstrucción de la memoria de control, se transfiere al *registro intermedio de control*. La parte izquierda de ese registro (*ver* Figura 17.1a) se conecta a las líneas de control que salen de la unidad de control. De este modo, *leer* una microinstrucción de la memoria de control es lo mismo que *ejecutar* la microinstrucción. El tercer elemento que muestra la figura es una unidad de secuenciamiento que carga el registro de dirección de control y emite una orden de lectura.

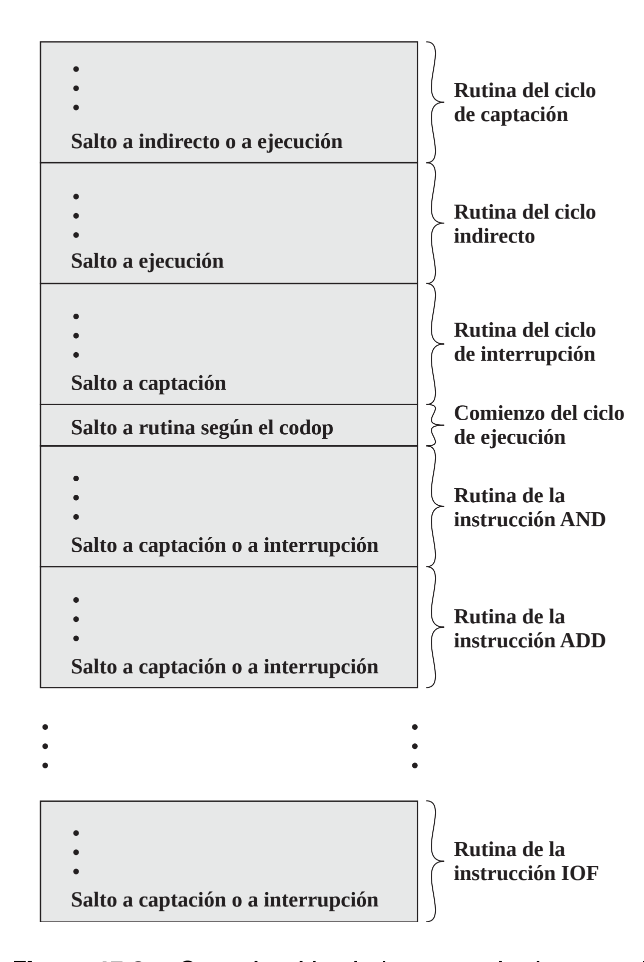


Figura 17.2. Organización de la memoria de control.

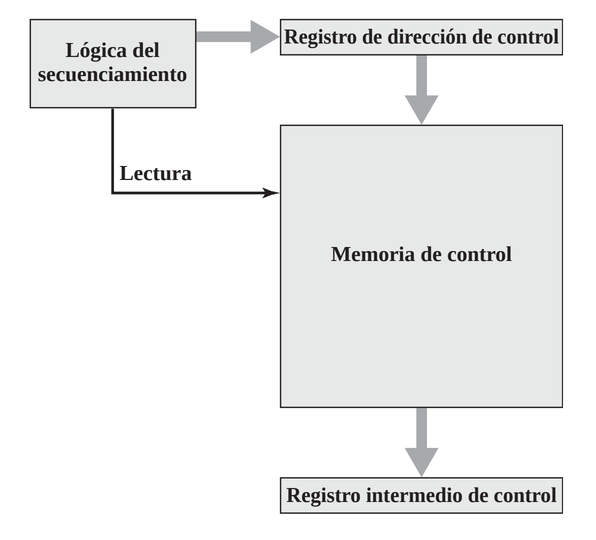


Figura 17.3. Microarquitectura de una unidad de control.

Examinemos esta estructura con mayor detalle, como representa la Figura 17.4. Comparándola con la Figura 16.4, vemos que la unidad de control sigue teniendo las mismas entradas (IR, indicadores de la ALU, reloj) y salidas (señales de control). La unidad de control funciona como sigue:

1. Para ejecutar una instrucción, la unidad lógica de secuenciamiento emite una orden de lectura a la memoria de control.
2. La palabra cuya dirección se especifica en el registro de dirección de control se lee en el registro intermedio de control.
3. El registro intermedio de control genera las señales de control y la información de dirección siguiente para la unidad lógica de secuenciamiento.
4. La unidad lógica de secuenciamiento carga en el registro de dirección de control una nueva dirección, basada en la información de dirección siguiente del registro intermedio de control y en los indicadores de la ALU.

Todo esto sucede durante un pulso de reloj.

El último paso recién mencionado requiere cierta elaboración. Al final de la ejecución de cada microinstrucción, la unidad lógica de secuenciamiento carga una nueva dirección en el registro de

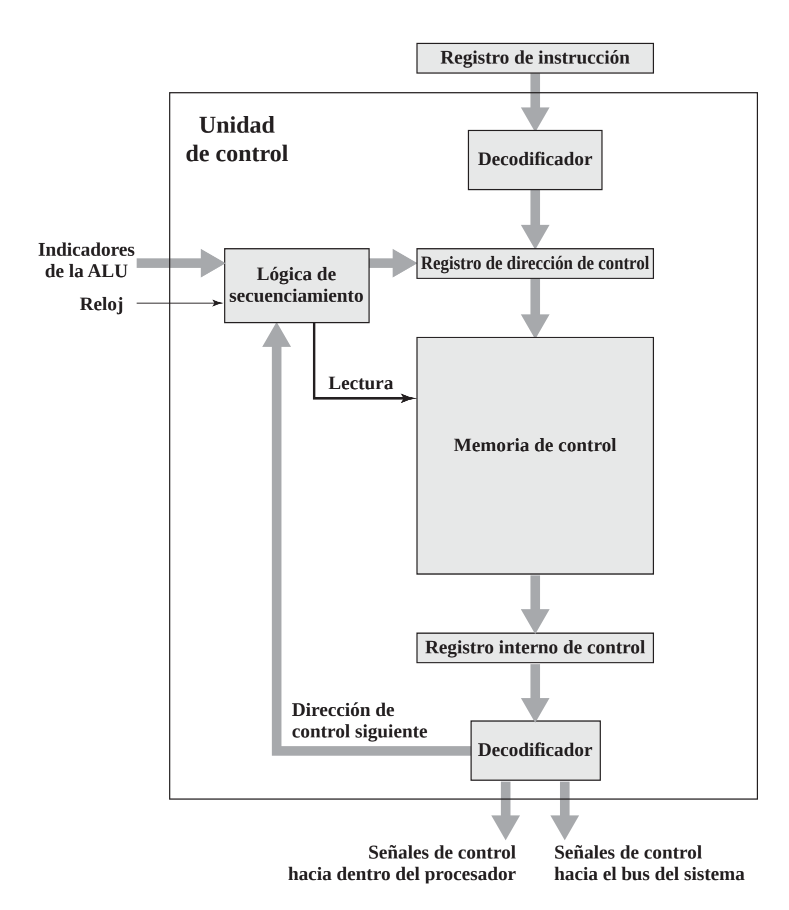


Figura 17.4. Funcionamiento de una unidad de control microprogramada.

dirección de control. Dependiendo del valor de los indicadores de la ALU y del registro intermedio de control, se toma una de las tres siguientes decisiones:

- **Captar la microinstrucción siguiente:** se suma 1 al registro de dirección de control.
- **Saltar a una nueva rutina según indica una microinstrucción de salto:** el campo de dirección del registro intermedio de control se carga en el registro de dirección de control.
- **Saltar a la rutina de una instrucción máquina:** se carga el registro de dirección de control en función del código de operación almacenado en IR.

La Figura 17.4 muestra dos módulos designados como *decodificador*. El decodificador de arriba traduce el código de operación de IR en una dirección de memoria de control. El decodificador de abajo no se usa con microinstrucciones horizontales pero sí con **microinstrucciones verticales** (Figura 17.1b). Como se mencionó, en una microinstrucción horizontal cada bit del campo de control corresponde a una línea de control. En una microinstrucción vertical se usa un código para cada acción a realizar —por ejemplo, MAR ((PC)—, y el decodificador traduce este código a señales de control individuales. La ventaja de las microinstrucciones verticales es que son más compactas (ocupan menos bits) que las microinstrucciones horizontales, a costa de añadir una pequeña lógica y cierto retardo temporal.

CONTROL DE WILKES

Como se ha mencionado, Wilkes fue el primero que propuso la utilización de una unidad de control microprogramada en 1951 [WILK51]. Más tarde elaboró su propuesta en un diseño más detallado [WILK53]. Es instructivo examinar esta propuesta inicial.

La configuración propuesta por Wilkes se representa en la Figura 17.5. El núcleo del sistema es una matriz parcialmente llena de diodos. Durante un ciclo máquina, se activa una fila de la matriz mediante un pulso. Esto produce señales en aquellos puntos en los que un diodo está presente (indicados mediante un punto en el diagrama). La primera parte de la fila genera las señales de control que gobiernan el funcionamiento del procesador. La segunda parte genera la dirección de la fila que será seleccionada mediante un pulso en el siguiente ciclo máquina. Por tanto, cada fila de la matriz es una microinstrucción y el trazado de la matriz es la memoria de control.

Al comienzo de un ciclo, la dirección de la fila a seleccionar está almacenada en el registro I. Esta dirección es la entrada del decodificador, el cual, cuando se activa mediante un pulso de reloj, selecciona una fila de la matriz. Dependiendo de las señales de control, durante el ciclo se pasa al registro II bien el código de operación almacenado en el registro de instrucción, o bien la segunda parte de la fila activada. El contenido del registro II se lleva entonces al registro I mediante un pulso de reloj. Se usan pulsos de reloj alternos para activar una fila de la matriz y para transferir el contenido del registro II al registro I. La configuración con dos registros es necesaria ya que el decodificador es sencillamente un circuito combinacional; con un único registro, la salida se convertiría en entrada dentro del mismo ciclo, originando un estado inestable.

Este esquema es muy similar al enfoque de microprogramación horizontal descrito anteriormente (Figura 17.1a). La principal diferencia es esta: en aquella descripción, el registro de dirección de control podía incrementarse en uno para acceder a la siguiente dirección. En el esquema de Wilkes, la dirección siguiente está contenida en la microinstrucción. Para permitir bifurcaciones, una fila debe

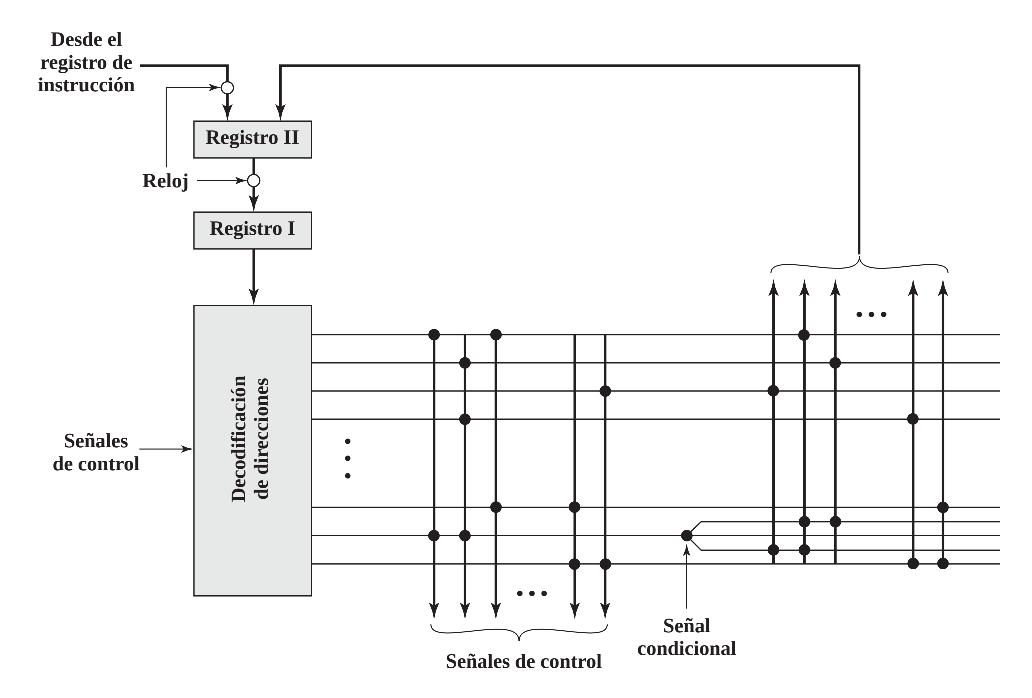


Figura 17.5. Unidad de control microprogramada de Wilkes.

contener dos partes de direcciones, controladas por una señal condicional (por ejemplo, un indicador), como muestra la figura.

Después de proponer este esquema, Wilkes proporciona un ejemplo de su utilización para implementar la unidad de control de una máquina sencilla. Este ejemplo, el primer diseño conocido de un procesador microprogramado, merece repetirse aquí porque ilustra muchos de los principios contemporáneos de la microprogramación.

El procesador de la máquina hipotética incluye los siguientes registros:

A multiplicando

B acumulador (mitad menos significativa)

C acumulador (mitad más significativa)

D registro de desplazamiento

Además, hay tres registros y dos indicadores de un bit accesibles solo por la unidad de control. Los registros son los siguientes:

E sirve como registro de dirección de memoria (MAR) y como almacenamiento temporal

F contador de programa

G otro registro temporal, usado en cálculos

La Tabla 17.1 enumera el conjunto de instrucciones máquina para este ejemplo. La Tabla 17.2 contiene el conjunto completo de microinstrucciones, expresadas de forma simbólica, que implementa

Tabla 17.1. Conjunto de instrucciones máquina del ejemplo de Wilkes.

Orden	Efecto de la orden
An	$C(Ac) + C(n)$ al Ac_1
Sn	$C(Ac) - C(n)$ al Ac_1
Hn	$C(n)$ al Ac_2
Vn	$C(Ac_2) \times C(n)$ al Ac_1, donde $C(n) \geq 0$
Tn	$C(Ac_1)$ a n, 0 al Ac
Un	$C(Ac_1)$ a n
Rn	$C(Ac) \times 2^{-(n+1)}$ al Ac
Ln	$C(Ac) \times 2^{n+1}$ al Ac
Gn	Si $C(Ac) < 0$, transferir el control a n; si $C(Ac) \geq 0$, ignorar (es decir, continuar secuencialmente)
In	Leer el siguiente carácter del mecanismo de entrada en n
On	Enviar $C(n)$ al mecanismo de salida

Notación: Ac = acumulador
Ac_1 = mitad más significativa del acumulador
Ac_2 = mitad menos significativa del acumulador
n = posición de memoria n
$C(X)$ = contenido de X (X = registro o posición de almacenamiento)

la unidad de control. Solo es necesario un total de 38 microinstrucciones para definir el sistema completamente.

La primera columna contiene la dirección (número de fila) de cada microinstrucción. Las direcciones referentes a códigos de operación están etiquetadas. De este modo, cuando se encuentra el código de operación de la instrucción de suma (A), se ejecuta la microinstrucción de la posición 5. Las columnas 2 y 3 expresan las acciones a realizar por la ALU y por la unidad de control, respectivamente. Cada expresión simbólica ha de traducirse en un conjunto de señales de control (bits de la microinstrucción). Las columnas 4 y 5 tienen que ver con la modificación y el uso de los dos indicadores (biestables). La columna 4 especifica la señal que ajusta el indicador. Por ejemplo, (1)C_s significa que el indicador número 1 se ajusta según el bit de signo del número contenido en el registro C. Si la columna 5 contiene un identificador de indicador, las columnas 6 y 7 contienen las dos direcciones de microinstrucción alternativas. Si no, la columna 6 especifica la dirección de la siguiente microinstrucción a captar.

Las instrucciones de la 0 a la 4 constituyen el ciclo de captación. La microinstrucción 4 presenta el código de operación a un decodificador, que genera la dirección de la microinstrucción a captar correspondiente a la instrucción máquina en curso. El lector debería ser capaz de deducir el funcionamiento completo de la unidad de control a partir de un cuidadoso estudio de la Tabla 17.2.

VENTAJAS E INCONVENIENTES

La ventaja principal que aporta el uso de la microprogramación para implementar una unidad de control es que simplifica su diseño. Por consiguiente, su implementación resulta más barata y menos

Tabla 17.2. Microinstrucciones del ejemplo de Wilkes.

Notación: *A, B, C...* representan los diversos registros de las unidades aritmética y de registros de control. *C* a *D* indica que los circuitos de conmutación conectan la salida del circuito *C* a la entrada del registro *D; (D* + A*)* a *C* indica que la salida del registro *A* se conecta a una entrada de la unidad de suma (la salida de *D* está conectada permanentemente a la otra entrada), y la salida del sumador al registro *C*. Un símbolo numérico *n* entre comillas (es decir, '*n*') representa la fuente cuya salida es el número *n* en unidades del dígito menos significativo.

		Unidad aritmética	Unidad de registros de control	Biestable condicional		Microinstrucción siguiente	
				Ajuste	Uso	0	1
	0		*F* a *G* y *E*			1	
	1		(*G* + '1') a *F*			2	
	2		Mem. a *G*			3	
	3		*G* a *E*			4	
	4		*E* a decodif.			—	
A	5	*C* a *D*				16	
S	6	*C* a *D*				17	
H	7	Mem. a *B*				0	
V	8	Mem. a *A*				27	
T	9	*C* a Mem.				25	
U	10	*C* a Mem.				0	
R	11	*B* a *D*	*E* a *G*			19	
L	12	*C* a *D*	*E* a *G*			22	
G	13		*E* a *G*	(1)C_s		18	
I	14	Entrada a Mem.				0	
O	15	Mem. a salida				0	
	16	(*D* + Mem.) a *C*				0	
	17	(*D* – Mem.) a *C*				0	
	18				1	0	1
	19	*D* a *B* (*R*)*	(*G* – '1') a *E*			20	
	20	*C* a *D*		(1)E_s		21	
	21	*D* a *C* (*R*)			1	11	0
	22	*D* a *C* (*L*)†	(*G* – '1') a *E*			23	
	23	*B* a *D*		(1)E_s		24	
	24	*D* a *B*				12	0
	25	'0' a *B*				26	
	26	*B* a *C*				0	
	27	'0' a *C*	'18' a *E*			28	
	28	*B* a *D*	*E* a *G*	(1)B_1		29	

(Continúa)

Tabla 17.2. Microinstrucciones del ejemplo de Wilkes *(continuación).*

	Unidad aritmética	Unidad de registros de control	Biestable condicional		Microinstrucción siguiente	
			Ajuste	Uso	0	1
29	*D* a *B* (*R*)	(*G* – '1') a *E*			30	
30	*C* a *D* (*R*)		(2)E_s	1	31	32
31	*D* a *C*			2	28	33
32	(*D* + *A*) a *C*			2	28	33
33	*B* a *D*		(1)B_1		34	
34	*D* a *B* (*R*)				35	
35	*C* a *D* (*R*)			1	36	37
36	*D* a *C*				0	
37	(*D* – *A*) a *C*				0	

* Desplazamiento a la derecha (*Right shift*). Los circuitos de conmutación de la unidad aritmética están organizados de tal forma que el dígito menos significativo del registro *C* se lleva a la posición más significativa del registro *B* durante las microoperaciones de desplazamiento a la derecha, y el dígito más significativo del registro *C* (dígito de signo) se repite (realizándose. por tanto, la corrección para números negativos).

† Desplazamiento a la izquierda (*Left shift*). Los circuitos de conmutación están dispuestos de manera similar para pasar el dígito más significativo del registro *B* a la posición menos significativa del registro *C* durante las microoperaciones de desplazamiento a la izquierda.

propensa a errores. Una unidad de control *cableada* contendrá lógica compleja para hacer el secuenciamiento a través de las muchas microoperaciones del ciclo de instrucción. Por otra parte, los decodificadores y la unidad lógica de secuenciamiento de una unidad de control microprogramada son elementos lógicos muy sencillos.

El principal inconveniente de una unidad microprogramada es que será algo más lenta que una unidad cableada de tecnología comparable. A pesar de ello, la microprogramación es la técnica dominante para implementar unidades de control en las arquitecturas CISC puras, debido a su facilidad de implementación. Los procesadores RISC, dado que tienen un formato de instrucción más sencillo, emplean normalmente unidades de control cableadas. Examinaremos ahora con más detalle la solución microprogramada.

17.2. SECUENCIAMIENTO DE MICROINSTRUCCIONES

Las dos tareas básicas realizadas por una unidad de control microprogramada son las siguientes:

- **Secuenciamiento de microinstrucciones:** obtener la siguiente microinstrucción de la memoria de control.
- **Ejecución de microinstrucciones:** generar las señales de control necesarias para ejecutar la microinstrucción.

Al diseñar una unidad de control, las dos tareas deben considerarse conjuntamente, ya que ambas afectan al formato de la microinstrucción y a la temporización de la unidad de control. En esta

sección, nos centraremos en el secuenciamiento y hablaremos lo mínimo posible sobre los temas de formato y temporización. Estos temas se examinarán más detalladamente en la sección siguiente.

CONSIDERACIONES RESPECTO AL DISEÑO

Hay dos cuestiones involucradas en el diseño de una técnica de secuenciamiento de microinstrucciones: el tamaño de la microinstrucción y el tiempo de generación de la dirección. El primer asunto es evidente: minimizar el tamaño de la memoria de control reduce el coste de este componente. El segundo asunto es sencillamente un deseo de ejecutar las microinstrucciones tan rápido como sea posible.

Cuando se ejecuta un microprograma, la dirección de la siguiente microinstrucción a ejecutar está en una de estas situaciones:

- Viene determinada por el registro de instrucción.
- Es la siguiente dirección secuencial.
- Es el destino de un salto.

La primera situación tiene lugar solo una vez por ciclo de instrucción, justo tras la captación de la instrucción. La segunda situación es la más común en la mayoría de los diseños. No obstante, el diseño no se puede optimizar solo para los accesos secuenciales. Los saltos, tanto condicionales como incondicionales, son una parte necesaria del microprograma. Además, las secuencias de microinstrucciones tienden a ser cortas; una de cada tres o cuatro microinstrucciones podría ser un salto [SIEW82]. Por consiguiente, es importante diseñar técnicas compactas y eficientes en cuanto al tiempo para los saltos a microinstrucciones.

TÉCNICAS DE SECUENCIAMIENTO

A partir de la microinstrucción en curso, de los indicadores de condición, y del contenido del registro de instrucción, hay que generar una dirección de la memoria de control para la siguiente microinstrucción. Se han usado numerosas técnicas. Podemos agruparlas en tres categorías, como ilustran las Figuras 17.6 a 17.8. Estas categorías se basan en el formato de la información de dirección de la microinstrucción:

- Dos campos de dirección.
- Un único campo de dirección.
- Formato variable.

La técnica más sencilla es tener dos campos de dirección en cada microinstrucción. La Figura 17.6 indica cómo se va a usar esta información. Se tiene un multiplexor que sirve de destino de los dos campos de dirección y del registro de instrucción. Basándose en la entrada de selección de dirección, el multiplexor transmite el código de operación o una de las dos direcciones al registro de dirección de control (*control address register*, CAR). El CAR se decodifica a continuación para producir la dirección de la siguiente microinstrucción. Las señales de selección de dirección son suministradas por un módulo de lógica de salto, cuyas entradas son los indicadores de la unidad de control y ciertos bits de la parte de control de la microinstrucción.

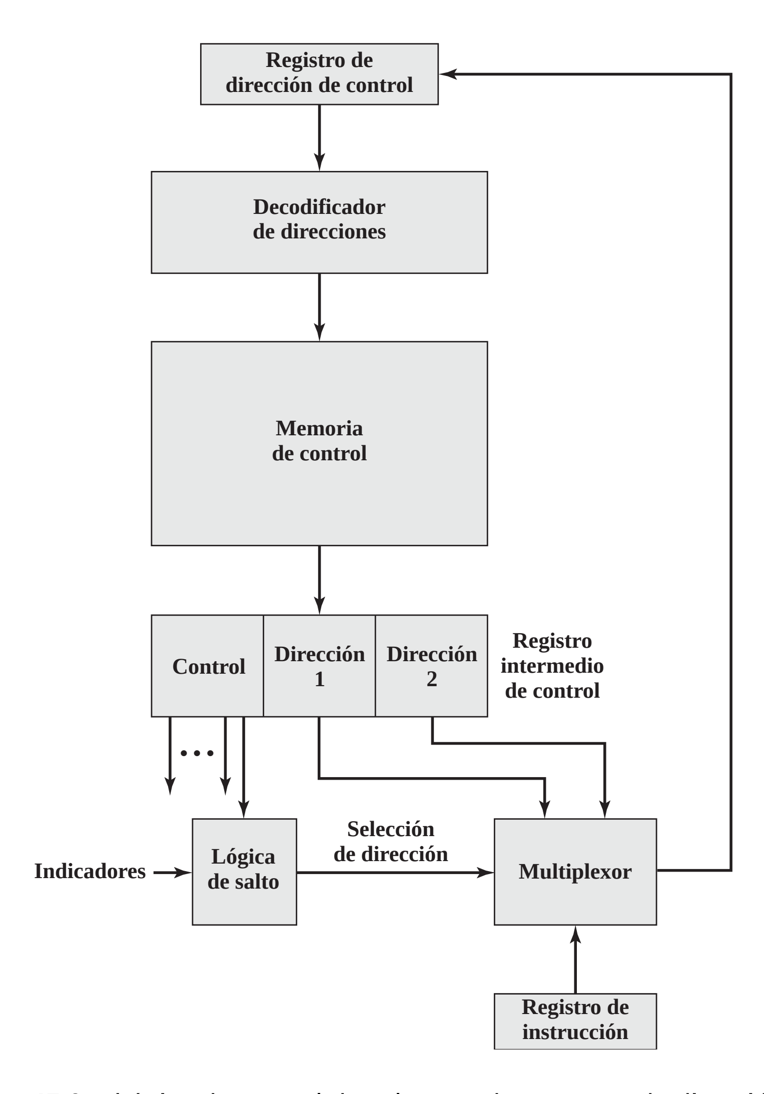


Figura 17.6. Lógica de control de salto con dos campos de dirección.

Aunque el método de dos direcciones es sencillo, necesita más bits por microinstrucción que las otras técnicas. Con alguna lógica adicional, se puede conseguir cierto ahorro. Una aproximación frecuente es tener un único campo de dirección (Figura 17.7). Con este enfoque, las opciones para la dirección siguiente son:

- Campo de dirección.
- Código del registro de instrucción.
- Siguiente dirección secuencial.

Las señales de selección de dirección determinan qué opción se escoge. Esta técnica reduce el número de campos de dirección a uno. Observe, sin embargo, que el campo de dirección a menudo no se usa. Por tanto, hay cierta ineficiencia en este esquema de codificación.

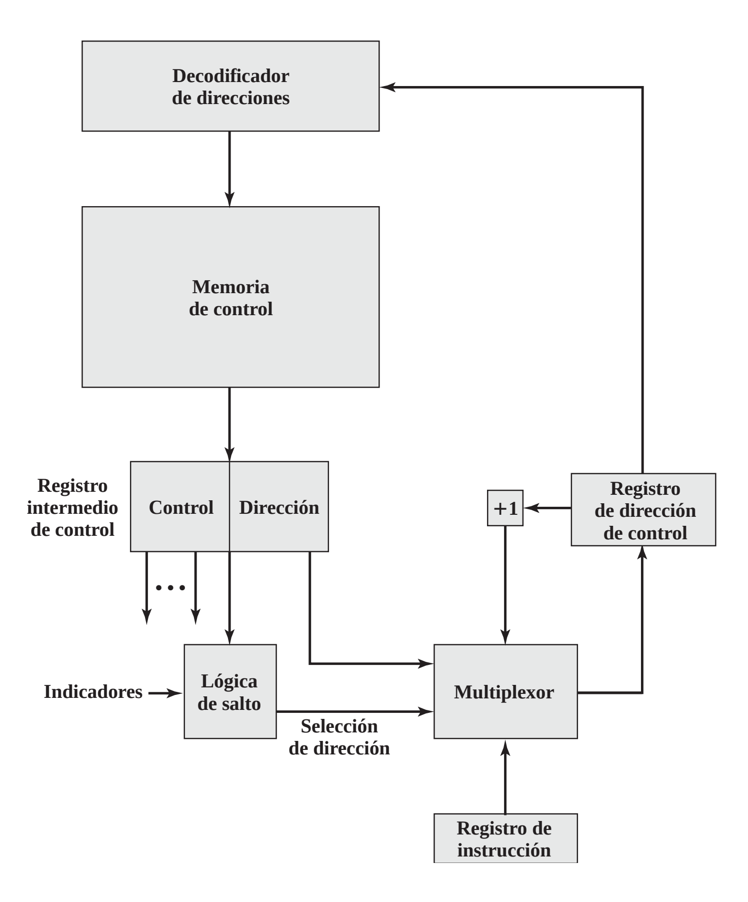


Figura 17.7. Lógica de control de salto con un único campo de dirección.

Otro método es proporcionar dos formatos de microinstrucción totalmente diferentes (Figura 17.8). Un bit designa qué formato se utilizará. En uno de los dos formatos, los demás bits se usan para activar señales de control. En el otro formato, algunos bits controlan el módulo de lógica de salto y los bits restantes suministran la dirección. En el primer formato, la dirección siguiente es la siguiente dirección secuencial o una dirección derivada del registro de instrucción. En el segundo formato, se especifica un salto condicional o incondicional. Un inconveniente de esta aproximación, tal como se ha descrito, es que se consume un ciclo completo por cada microinstrucción de salto. Con las otras técnicas, la generación de la dirección sucede como parte del mismo ciclo en el que se generan las señales de control, lo cual minimiza los accesos a la memoria de control.

Las aproximaciones que se acaban de describir son generales. Las implementaciones específicas con frecuencia usarán una variación o una combinación de estas técnicas.

GENERACIÓN DE DIRECCIONES

Hemos enfocado el problema del secuenciamiento desde el punto de vista de las consideraciones sobre el formato y de los requisitos de lógica en general. Otro punto de vista es considerar las diversas formas de obtener o calcular la siguiente dirección.

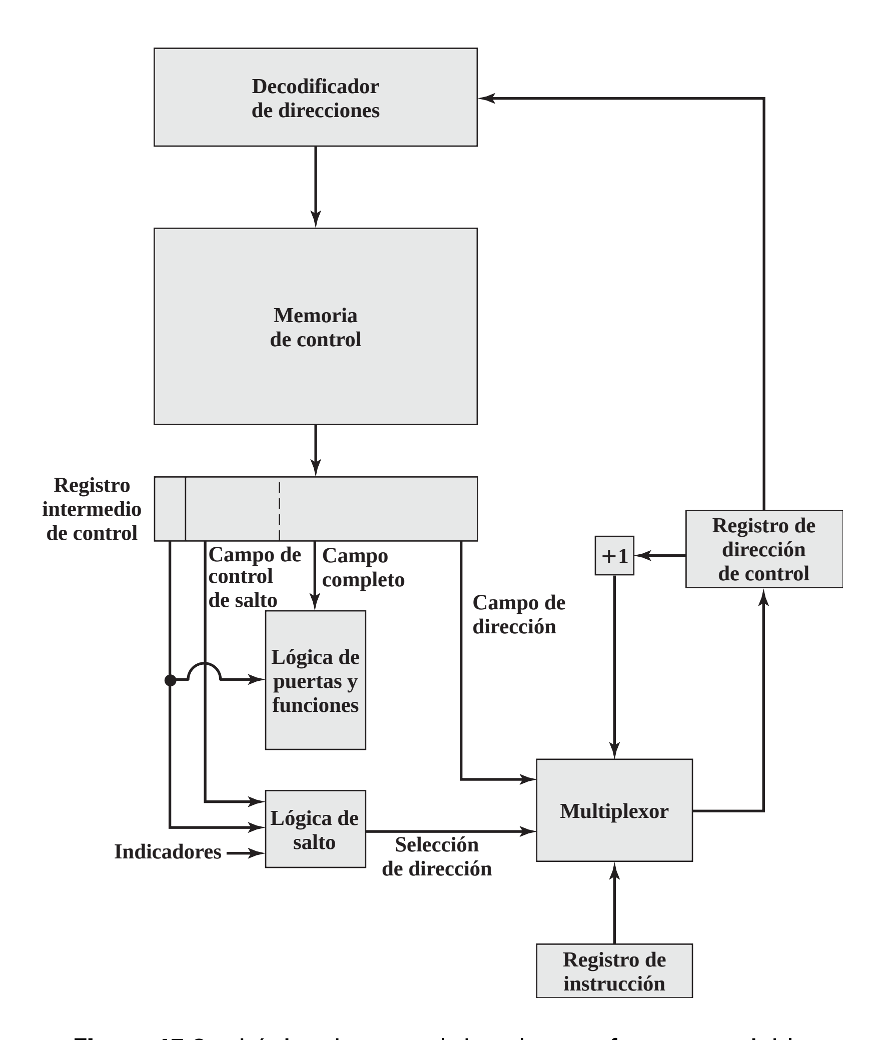


Figura 17.8. Lógica de control de salto con formato variable.

La Tabla 17.3 relaciona las diversas técnicas de generación de la dirección. Estas se pueden dividir en técnicas explícitas, en las que la dirección aparece explícitamente en la microinstrucción, y técnicas implícitas, que requieren lógica adicional para generar la dirección.

Nos hemos ocupado esencialmente de las técnicas explícitas. Con un enfoque de dos campos, hay dos direcciones alternativas disponibles en cada microinstrucción. Usando un único campo de

Tabla 17.3. Técnicas de generación de direcciones de microinstrucción.

Explícitas	Implícitas
Dos campos	Traducción
Salto incondicional	Adición
Salto condicional	Control residual

dirección o un formato variable, se pueden implementar varias instrucciones de salto. Una instrucción de salto condicional depende de los siguientes tipos de información:

- Indicadores de la ALU.
- Parte del código de operación o campos de modo de direccionamiento de la instrucción máquina.
- Partes de un registro seleccionado, tales como el bit de signo.
- Bits de estado dentro de la unidad de control.

También se usan frecuentemente algunas técnicas implícitas. Una de ellas, la traducción, se necesita en casi todos los diseños. La parte de una instrucción máquina que contiene el código de operación se traduce a una dirección de microinstrucción. Esto ocurre solo una vez por ciclo de instrucción.

Una técnica implícita habitual consiste en combinar o sumar dos partes de una dirección para formar la dirección completa. Este procedimiento fue adoptado por la familia IBM S/360 [TUCK67] y usado por muchos de los modelos S/370. Usaremos el IBM 3033 como ejemplo.

El registro de dirección de control del IBM 3033, de trece bits, se ilustra en la Figura 17.9. Se pueden distinguir dos partes en la dirección. Los ocho bits más significativos (00-07) no cambian normalmente de un ciclo de microinstrucción al siguiente. Durante la ejecución de una microinstrucción, estos ocho bits se copian directamente desde un campo de ocho bits de la microinstrucción (el campo BA) en los ocho bits más significativos del registro de dirección de control. Ello define un bloque de 32 microinstrucciones en la memoria de control. Los otros cinco bits del registro de dirección de control se ajustan para especificar la dirección concreta de la microinstrucción a captar a continuación. Cada uno de estos bits viene determinado por un campo de cuatro bits (excepto uno por un campo de siete bits) de la microinstrucción en curso; cada campo especifica la condición para ajustar el bit correspondiente. Por ejemplo, un bit del registro de dirección de control puede ponerse a 1 o a 0 dependiendo de si se produjo acarreo en la última operación de la ALU.

La última técnica de la lista de la Tabla 17.3 se denomina *control residual*. Esta aproximación implica el uso de una dirección de microinstrucción guardada previamente en un almacenamiento temporal dentro de la unidad de control. Por ejemplo, algunos conjuntos de microinstrucciones están dotados de la posibilidad de hacer llamadas a subrutinas. Un registro interno o una pila de registros se usan para guardar las direcciones de retorno. Un ejemplo de esta técnica aparece en el LSI-11, que examinamos ahora.

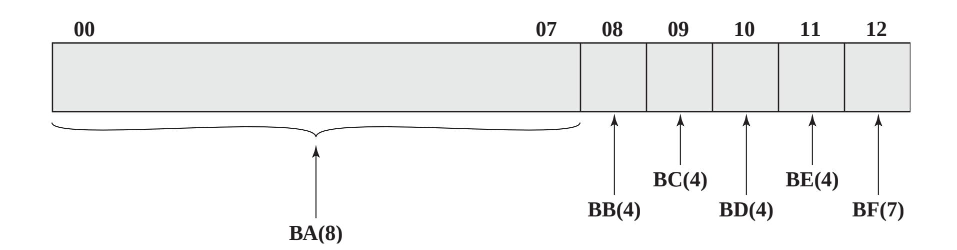


Figura 17.9. Registro de dirección de control del IBM 3033.

SECUENCIAMIENTO DE MICROINSTRUCCIONES EN EL LSI-11

El microcomputador LSI-11 es una versión del PDP-11 con los componentes principales del sistema en una misma tarjeta. El LSI-11 está implementado usando una unidad de control microprogramada [SEBE76].

El LSI-11 utiliza microinstrucciones de 22 bits y una memoria de control de 2Kpalabras de 22 bits. La dirección de la siguiente microinstrucción se determina de una de estas cinco formas:

- **Dirección secuencial siguiente:** en ausencia de otras instrucciones, el registro de dirección de control de la unidad de control se incrementa en 1.
- **Traducción del código de operación:** al comienzo de cada ciclo de instrucción, la dirección de la siguiente microinstrucción viene determinada por el código de operación.
- **Llamada/retorno de subrutina:** explicada más abajo.
- **Comprobación de interrupciones:** ciertas microinstrucciones especifican una comprobación de interrupciones. Si ha ocurrido una interrupción, ello determina la dirección de la siguiente microinstrucción.
- **Salto:** se usan microinstrucciones de salto condicional e incondicional.

Se proporciona la posibilidad de subrutinas de un nivel. Un bit de cada microinstrucción se dedica a esta tarea. Cuando el bit está a uno, un registro de retorno de once bits se carga con el contenido actualizado del registro de dirección de control. Una instrucción posterior que especifique un retorno hará que se cargue el registro de dirección de control con el contenido del registro de retorno.

El retorno es una forma de microinstrucción de salto incondicional. Otra forma de salto incondicional hace que se carguen los bits del registro de dirección de control con once bits de la microinstrucción. La microinstrucción de salto condicional hace uso de un código de comprobación de cuatro bits dentro de la microinstrucción. Este código especifica la comprobación de diversos códigos de condición de la ALU para determinar la decisión de salto. Si la condición no es cierta, se escoge la siguiente dirección secuencial. Si es cierta, los ocho bits menos significativos del registro de dirección de control se cargan con ocho bits de la microinstrucción. Esto permite el salto dentro de una página de memoria de 256 palabras.

Como puede observarse, el LSI-11 incluye un potente secuenciamiento de direcciones dentro de la unidad de control. Ello da al microprogramador una flexibilidad apreciable y puede facilitar la tarea de la microprogramación. Por otra parte, esta aproximación requiere más lógica en la unidad de control que otra con menores capacidades.

17.3. EJECUCIÓN DE MICROINSTRUCCIONES

El ciclo de microinstrucción es el evento básico de un procesador microprogramado. Cada ciclo de compone de dos partes: captación y ejecución. La parte de captación depende de la generación de una dirección de microinstrucción, que fue tratada en la sección precedente. Esta sección se ocupa de la ejecución de una microinstrucción.

Recordemos que el resultado de la ejecución de una microinstrucción es la generación de señales de control. Algunas de estas señales controlan puntos internos del procesador. Las demás señales van al bus de control externo o a otras interfaces externas. Como una función accesoria, se determina la dirección de la siguiente microinstrucción.

La descripción precedente sugiere la organización de la unidad de control que se muestra en la Figura 17.10. Esta versión ligeramente revisada de la Figura 17.4 subraya el centro de atención de esta sección. Los principales módulos de este diagrama ya deben estar claros. El módulo de lógica de secuenciamiento contiene la lógica que realiza las funciones estudiadas en la sección anterior. Genera la dirección de la siguiente microinstrucción, usando como entradas el registro de instrucción, los indicadores de la ALU, el registro de dirección de control (para incrementarlo) y el registro intermedio de control. El último puede proporcionar una dirección real, bits de control o ambos. Este módulo está controlado por un reloj que determina la temporización del ciclo de microinstrucción.

El módulo de lógica de control genera las señales de control en función de algunos de los bits de la microinstrucción. Debería quedar claro que el formato y el contenido de la microinstrucción determinarán la complejidad del módulo de lógica de control.

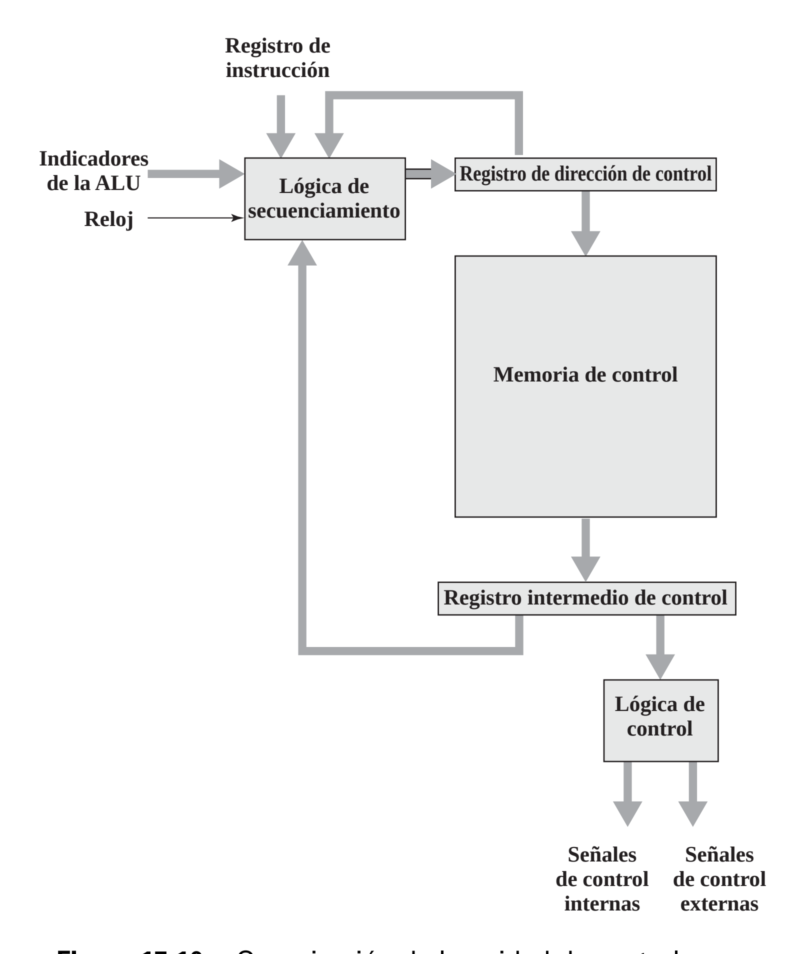


Figura 17.10. Organización de la unidad de control.

UNA TAXONOMÍA DE LAS MICROINSTRUCCIONES

Las microinstrucciones se pueden clasificar de varias formas. Las distinciones que generalmente se hacen en la bibliografía incluyen

- Vertical/horizontal.
- Empaquetada/no empaquetada.
- Microprogramación *hard*/*soft*.
- Codificación directa/indirecta.

Todas ellas se refieren al formato de la microinstrucción. Ninguno de estos términos se ha usado de una manera coherente y precisa en la bibliografía. No obstante, un examen de estas parejas de cualidades sirve para aclarar las diferentes alternativas en el diseño de las microinstrucciones. En los siguientes párrafos consideramos primero el principal asunto, referente al diseño, que subyace en todas estas parejas de características alternativas, y después consideraremos los conceptos que cada pareja de alternativas sugiere.

En la propuesta original de Wilkes [WILK51], cada bit de una microinstrucción producía directamente una señal de control o un bit de la dirección siguiente. Hemos visto, en la sección precedente, que son posibles esquemas de secuenciamiento de direcciones más complejos que usan menos bits por microinstrucción. Tales esquemas requieren un módulo de lógica de secuenciamiento más complejo. Existe un tipo de compromiso semejante para la parte de la microinstrucción que atañe a las señales de control. Se pueden ahorrar bits de la palabra de control codificando la información de control, y decodificándola más tarde para producir las señales de control.

¿Cómo puede hacerse esta codificación? Para responder a esta pregunta consideremos que hay un total de K señales de control internas y externas diferentes que tiene que generar la unidad de control. En el esquema de Wilkes se dedicarían a este propósito K bits de la microinstrucción. Esto permite que se puedan generar las 2^K combinaciones posibles de señales de control durante cualquier ciclo de instrucción. Pero somos capaces de hacerlo mejor si observamos que no todas las combinaciones posibles se usarán. Veamos algunos ejemplos:

- Dos fuentes no se pueden llevar al mismo destino (por ejemplo, C_2 y C_8 en la Figura 16.5).
- Un registro no puede ser a la vez fuente y destino (por ejemplo, C_5 y C_{12} en la Figura 16.5).
- Solo un patrón de señales de control se puede presentar a la ALU cada vez.
- Solo un patrón de señales de control se puede presentar al bus de control externo cada vez.

De este modo, para un procesador dado, puede hacerse una lista con todas las posibles combinaciones de señales de control admisibles, obteniendo un número de posibilidades $Q < 2^K$ que podrían codificarse con $\log_2 Q$ bits, siendo $(\log_2 Q) < K$. Esta sería la forma más estricta posible de codificación que preserva todas las combinaciones permisibles de señales de control. En la práctica, este sistema de codificación no se usa, por dos razones:

- Es tan difícil de programar como un esquema decodificado puro (como el de Wilkes). Este punto se discutirá más a fondo dentro de poco.
- Requiere un módulo de lógica de control complejo y, por consiguiente, lento.

En lugar de eso, se adoptan algunos compromisos. Los hay de dos tipos:

- Se usan más bits de los estrictamente necesarios para codificar las posibles combinaciones.
- Algunas combinaciones que son físicamente permisibles no se pueden codificar.

El último tipo de compromiso tiene el efecto de reducir el número de bits. El resultado neto, sin embargo, es que se usan más bits que $\log_2 Q$.

En la siguiente subsección discutiremos técnicas de codificación específicas. El resto de esta subsección se ocupa de los efectos de la codificación y de los diversos términos usados para describirla.

Basándonos en lo anterior, podemos ver que el campo de señales de control del formato de microinstrucción se encuadra dentro de un espectro. En un extremo, hay un bit para cada señal de control; en el otro extremo, se usa un formato muy codificado. La Tabla 17.4 muestra otras características de una unidad de control microprogramada que también se encuadran dentro de espectros de posibilidades y que, por lo general, dependen del espectro del grado de codificación.

La segunda pareja de características de la tabla es bastante evidente. El esquema puro de Wilkes es el que requiere más bits. También debería estar claro que este extremo ofrece la visión más detallada del hardware. El microprogramador maneja individualmente cada señal de control. La codificación se hace de tal modo que se agrupan funciones o recursos, y debido a ello el microprogramador ve el procesador a un nivel más alto, menos detallado. Además, la codificación se diseña para facilitar la microprogramación. De nuevo, debería quedar claro que la labor de comprender y orquestar el uso de todas las señales de control es difícil. Como se mencionó, una de las consecuencias típicas de la codificación es que impide el uso de ciertas combinaciones que serían permisibles si no existiera esta.

El párrafo precedente discute el diseño de la microinstrucción desde el punto de vista del microprogramador. Pero el grado de codificación también puede considerarse viendo sus efectos sobre el

Tabla 17.4. El espectro de las microinstrucciones.

Características	
Microinstrucción no codificada	Microinstrucción muy codificada
Muchos bits	Pocos bits
Visión detallada del hardware	Visión global del hardware
Difícil de programar	Fácil de programar
Concurrencia explotada completamente	Concurrencia no explotada completamente
Poca o ninguna lógica de control	Lógica de control compleja
Ejecución rápida	Ejecución lenta
Optimización de las prestaciones	Optimización de la programación
Terminología	
No empaquetada	Empaquetada
Horizontal	Vertical
Hard	*Soft*

hardware. Con un formato puro no codificado, se necesita poca o ninguna lógica de decodificación; cada bit genera una señal de control individual. Conforme se usan esquemas de codificación más compactos y globales, se necesita una lógica de decodificación más compleja. Esto puede afectar proporcionalmente a las prestaciones. Se necesita más tiempo para propagar las señales a través de las puertas de una lógica de control más compleja. Por tanto, la ejecución de microinstrucciones codificadas tarda más tiempo que la ejecución de las no codificadas.

De este modo, todas las características relacionadas en la Tabla 17.4 se distribuyen a lo largo de un espectro de estrategias de diseño. En general, un diseño que cae hacia el extremo izquierdo del espectro se propone optimizar las prestaciones de la unidad de control. Los diseños del extremo derecho están más interesados en optimizar el proceso de microprogramación. En efecto, los conjuntos de microinstrucciones cercanos al extremo derecho del espectro se parecen mucho a los conjuntos de instrucciones máquina. Un buen ejemplo de ello es el diseño del LSI-11, descrito más adelante en esta sección. Típicamente, cuando el objetivo es sencillamente implementar una unidad de control, el diseño estará cerca del extremo izquierdo del espectro. El diseño del IBM 3033, que se estudiará luego, está en esta categoría. Como veremos más adelante, algunos sistemas permiten que diferentes usuarios construyan microprogramas diferentes usando el mismo tipo de microinstrucción. En este segundo caso, el diseño caerá probablemente cerca del extremo derecho del espectro.

Podemos ocuparnos ahora de alguna terminología introducida anteriormente. La Tabla 17.4 indica cómo tres de estas parejas de términos se relacionan con el espectro de las microinstrucciones. Fundamentalmente, cualquiera de estas parejas describe la misma cosa pero da importancia a diferentes características de diseño.

El grado de empaquetamiento se relaciona con el grado de identificación entre una tarea de control determinada y algunos bits específicos de la microinstrucción. Cuanto más *empaquetados* están los bits, un número dado de bits contiene más información. Por lo tanto, el empaquetamiento se relaciona con la codificación. Los términos *horizontal* y *vertical* se relacionan con el ancho relativo de las microinstrucciones. [SIEW82] indica a modo de regla empírica que las microinstrucciones verticales tienen longitudes en el rango de 16 a cuarenta bits, y las microinstrucciones horizontales longitudes de cuarenta a cien bits. Los términos microprogramación *hard* y *soft* se utilizan para indicar el grado de proximidad a las señales de control subyacentes y a la configuración del hardware. Los microprogramas *hard* generalmente son fijos y se sitúan en memoria de sólo lectura. Los microprogramas *soft* son más cambiables y sugieren una microprogramación por parte del usuario.

La otra pareja de términos mencionados al comienzo de esta subsección se refiere a codificación directa frente a indirecta, un asunto al que volvemos ahora.

CODIFICACIÓN DE LAS MICROINSTRUCCIONES

En la práctica, las unidades de control microprogramadas no se diseñan utilizando un formato de microinstrucción puro no codificado u horizontal. Se hace uso, al menos, de algún grado de codificación para reducir el ancho de la memoria de control y simplificar la tarea de la microprogramación.

La técnica básica de codificación se ilustra en la Figura 17.11a. La microinstrucción se organiza como un conjunto de campos. Cada campo contiene un código que, tras la decodificación, activa una o más señales de control.

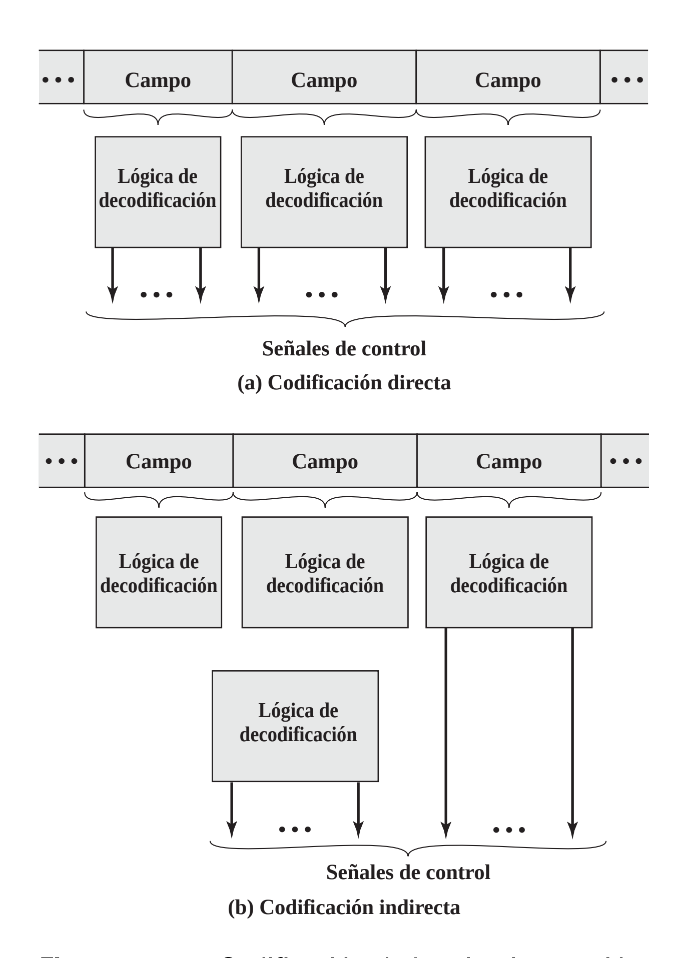


Figura 17.11. Codificación de la microinstrucción.

Consideremos las implicaciones de este esquema. Cuando la microinstrucción se ejecuta, cada campo se decodifica y genera señales de control. Así, con N campos, se especifican N acciones simultáneas. Cada acción se traduce en la activación de una o más señales de control. Generalmente, aunque no siempre, querremos diseñar el formato de modo que cada señal de control no pueda ser activada por más de un campo. Claramente, no obstante, debe ser posible activar cada señal de control al menos por un campo.

Consideremos ahora un campo individual. Un campo que conste de L bits puede contener uno de los 2^L códigos posibles, cada uno de los cuales puede codificar un patrón diferente de señales de control. Como solo puede aparecer un código en un campo en un momento dado, los códigos son mutuamente exclusivos y, por lo tanto, las acciones que ellos producen también lo son.

El diseño de un formato de microinstrucción codificado puede plantearse ahora en términos muy simples:

- Organizar el formato en campos independientes. Es decir, cada campo representa un conjunto de acciones (patrón de señales de control) de tal forma que pueden ocurrir simultáneamente acciones de diferentes campos.

- Definir cada campo de modo que las acciones alternativas que puede especificar ese campo sean mutuamente exclusivas. Es decir, que solo una de las acciones especificadas por un determinado campo pueda ocurrir en un momento dado.

Se pueden usar dos planteamientos para la organización de la microinstrucción codificada en campos: funcional y por recursos. El método de *codificación funcional* identifica funciones dentro de la máquina y designa los campos según el tipo de función. Por ejemplo, si se pueden utilizar varias fuentes para la transferencia de datos al acumulador, se puede utilizar un campo para este propósito, especificando cada código una fuente diferente. La *codificación por recursos* ve a la máquina como un conjunto de recursos independientes y le dedica un campo a cada uno de ellos (por ejemplo, E/S, memoria, ALU).

Otro aspecto de la codificación es si esta es directa o indirecta (Figura 17.11b). En la codificación indirecta, se utiliza un campo para determinar la interpretación de otro campo. Por ejemplo, consideremos una ALU capaz de realizar ocho operaciones aritméticas y ocho operaciones de desplazamiento diferentes. Se podría usar un campo de un bit para indicar si se trata de una operación aritmética o de desplazamiento, y un campo de tres bits podría indicar la operación. Generalmente esta técnica implica dos niveles de decodificación, incrementando el retardo de propagación de las señales.

La Figura 17.12 es un ejemplo sencillo de estos conceptos. Se supone un procesador con un único acumulador y varios registros internos, tales como un contador de programa y un registro temporal para la entrada de la ALU. La Figura 17.12a muestra un formato muy vertical. Los tres primeros bits indican el tipo de operación, los tres siguientes codifican la operación, y los dos últimos seleccionan un registro interno. La Figura 17.12b es una solución más horizontal, aunque todavía usa cierta codificación. En este caso, las funciones diferentes aparecen en campos diferentes.

EJECUCIÓN DE MICROINSTRUCCIONES EN EL LSI-11

El LSI-11 [SEBE76] es un buen ejemplo del planteamiento de microinstrucciones verticales. Veamos en primer lugar la organización de la unidad de control y después el formato de la microinstrucción.

Organización de la unidad de control del LSI-11. El LSI-11 es el primer miembro de la familia PDP-11 que se ofreció como procesador en una sola tarjeta. La tarjeta contiene tres circuitos LSI, un bus interno conocido como *bus de microinstrucciones* (*microinstruction bus*, MIB) y alguna lógica de interfaz adicional.

La Figura 17.13 representa, de forma simplificada, la organización del procesador LSI-11. Los tres circuitos son el de datos, el de control y el de memoria de control. El circuito de datos contiene una ALU de ocho bits, 26 registros de ocho bits y almacenamiento para varios códigos de condición. Dieciséis de esos registros se usan para implementar los ocho registros de uso general de 16 bits del PDP-11. Otros incluyen una palabra de estado del programa, un registro de dirección de memoria (MAR), y un registro intermedio de memoria. Como la ALU trata solo ocho bits a la vez, se requieren dos pasos a través de ella para implementar una operación aritmética de 16 bits del PDP-11. Esto lo controla el microprograma.

El circuito o circuitos de memoria de control contienen la memoria de control, de 22 bits de ancho. El circuito de control contiene la lógica de secuenciamiento y ejecución de las microinstrucciones. Contiene también el registro de dirección de control, el registro de datos de control y una copia del registro de instrucción máquina.

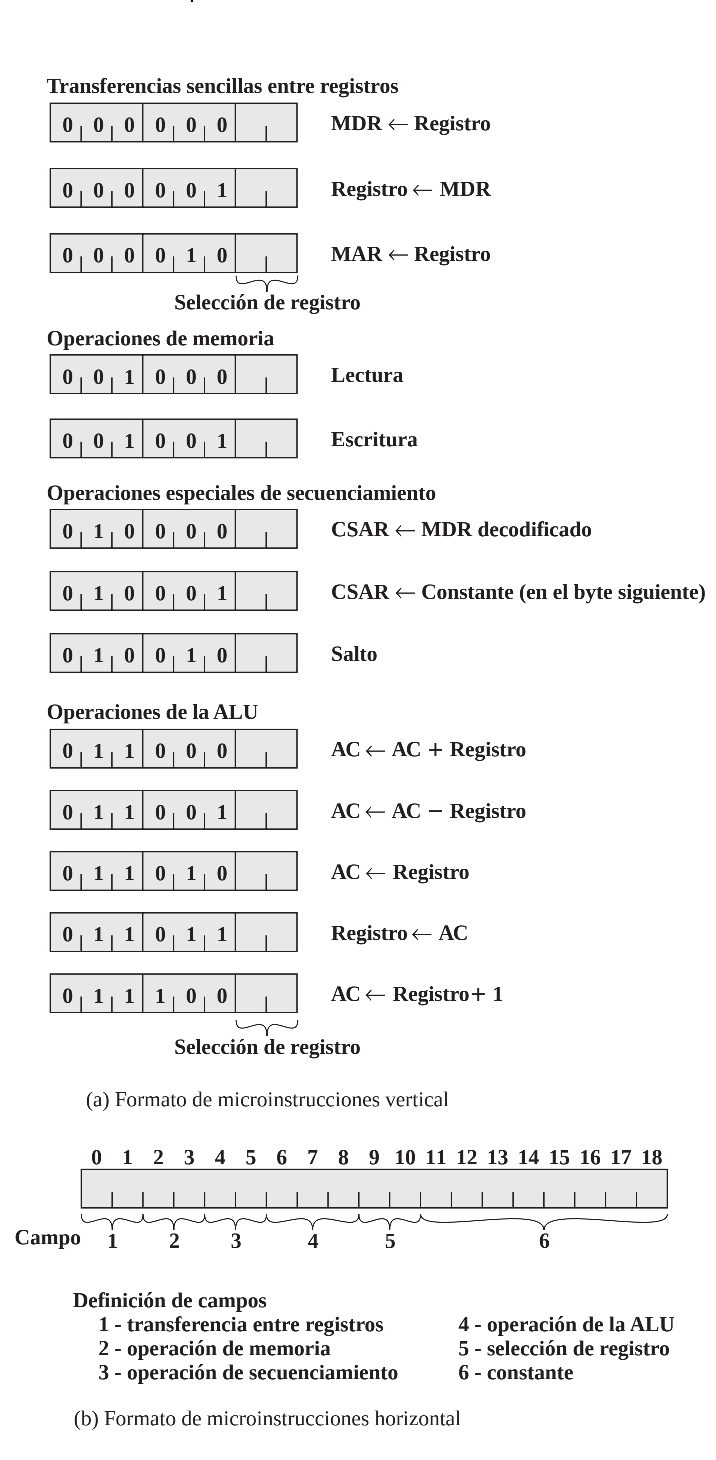


Figura 17.12. Formatos de microinstrucción alternativos para una máquina sencilla.

El MIB interconecta todos los componentes. Durante la captación de la microinstrucción, el circuito de control sitúa una dirección de once bits en el MIB. Se accede a la memoria de control para leer una microinstrucción de 22 bits, que se sitúa en el MIB. Los 16 bits menos significativos van

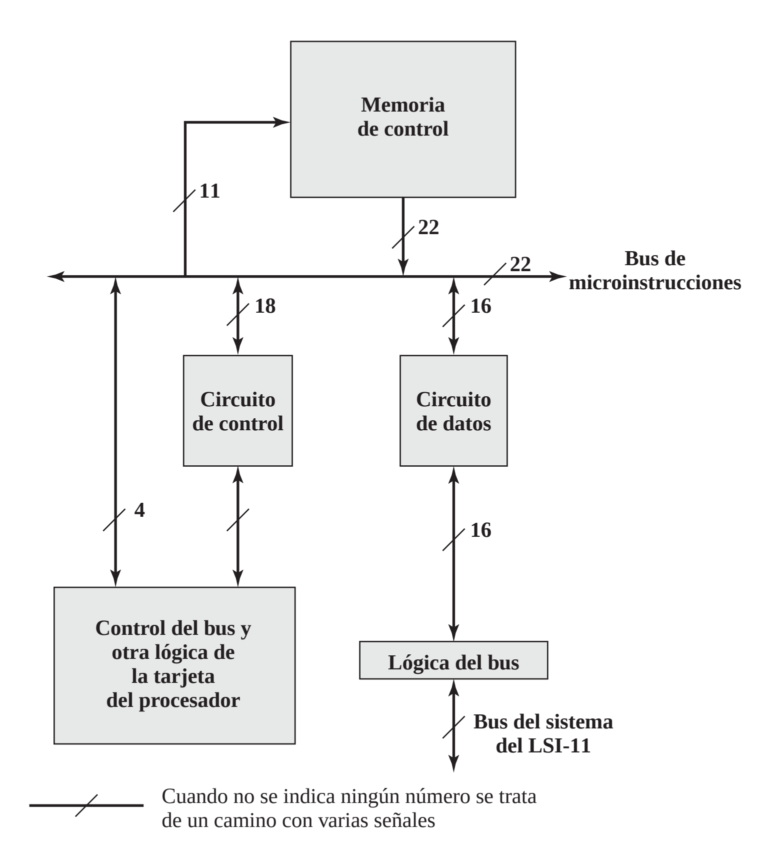


Figura 17.13. Diagrama de bloques simplificado del procesador LSI-11.

al circuito de datos, y los 18 bits menos significativos van al circuito de control. Los cuatro bits más significativos controlan funciones especiales de la tarjeta del procesador.

La Figura 17.14 ofrece una visión todavía simplificada, aunque más detallada, de la unidad de control del LSI-11: la figura no tiene en cuenta los límites de los circuitos. El esquema de secuenciamiento de direcciones descrito en la Sección 17.2 se implementa en dos módulos. El módulo de control de secuencia del microprograma suministra el control de secuencia global, y es capaz de incrementar el registro de dirección de microinstrucciones y de realizar saltos incondicionales. Las otras formas de cálculo de direcciones se llevan a cabo por una tabla de traducción independiente. Esta es un circuito combinacional que genera una dirección a partir de la microinstrucción, la instrucción máquina, el contador de programa de microinstrucciones y un registro de interrupción.

La tabla de traducción entra en juego en las siguientes ocasiones:

- El código de operación se utiliza para determinar el inicio de una microrrutina.
- En el momento apropiado, los bits de modo de direccionamiento de la microinstrucción se comprueban para realizar el direccionamiento oportuno.
- Las condiciones de interrupción se comprueban periódicamente.
- Se evalúan las microinstrucciones de salto condicional.

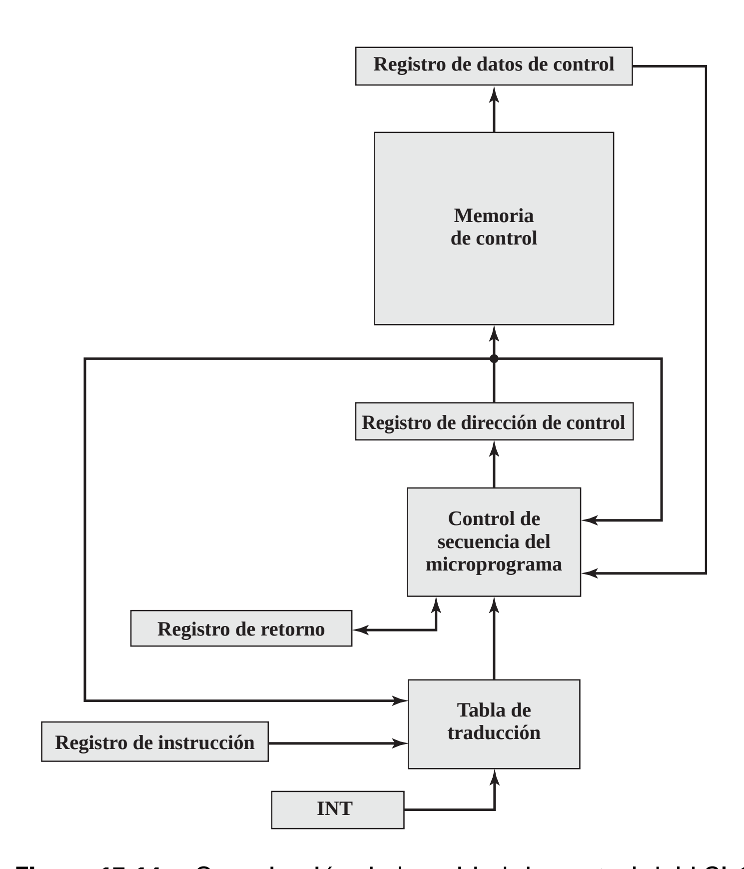


Figura 17.14. Organización de la unidad de control del LSI-11.

Formato de microinstrucción del LSI-11. El LSI-11 utiliza un formato de microinstrucción extremadamente vertical, con un ancho de solo 22 bits. El conjunto de microinstrucciones se parece mucho al conjunto de instrucciones máquina del PDP-11 que él implementa. Este diseño pretendía optimizar las prestaciones de la unidad de control con la restricción de un diseño vertical y fácilmente programable. La Tabla 17.5 contiene una lista de algunas microinstrucciones del LSI-11.

La Figura 17.15 muestra el formato de 22 bits de la microinstrucción del LSI-11. Los cuatro bits más significativos controlan funciones especiales en la tarjeta del procesador. El bit de traducción permite a la tabla de traducción comprobar si hay interrupciones pendientes. El bit de carga del registro de retorno se utiliza al final de una microrrutina para hacer que se use el contenido del registro de retorno como dirección de la siguiente microinstrucción.

Los 16 bits restantes se usan para microoperaciones muy codificadas. El formato se parece mucho al de una instrucción máquina, con un código de operación de longitud variable y uno o más operandos.

EJECUCIÓN DE MICROINSTRUCCIONES EN EL IBM 3033

La memoria de control estándar del IBM 3033 consta de 4K palabras. La primera mitad de ellas (0000-07FF) contiene microinstrucciones de 108 bits, mientras que las restantes (0800-0FFF) se utilizan para almacenar microinstrucciones de 126 bits. El formato se representa en la Figura 17.16. Aun

Tabla 17.5. Algunas microinstrucciones del LSI-11.

Operaciones aritméticas	Desplazar palabra (byte) a la derecha (izquierda) con (sin) acarreo
Sumar palabra (byte, literal)	Complementar palabra (byte)
Comprobar palabra (byte, literal)	**Operaciones generales**
Incrementar palabra (byte) en 1	Transferir palabra (byte)
Incrementar palabra (byte) en 2	Saltar
Negar palabra (byte)	Retornar
Incrementar (decrementar) byte condicionalmente	Saltar condicionalmente
Sumar palabra (byte) condicionalmente	Poner a uno (a cero) los indicadores
Sumar palabra (byte) con acarreo	Cargar G bajo
Sumar dígitos condicionalmente	Transferir palabra (byte) condicionalmente
Restar palabra (byte)	**Operaciones de Entrada/Salida**
Comparar palabra (byte, literal)	Captar palabra (byte)
Restar palabra (byte) con acarreo	Captar palabra (byte) de estado
Decrementar palabra (byte) en 1	Leer
Operaciones lógicas	Escribir
Y de palabra (byte, literal)	Leer (escribir) e incrementar palabra (byte) en 1
Comprobar palabra (byte)	Leer (escribir) e incrementar palabra (byte) en 2
O de palabra (byte)	Leer (escribir) reconocimiento
O exclusiva de palabra (byte)	Enviar palabra (byte, estado) a la salida
Poner a cero bit de palabra (byte)	

tratándose de un formato bastante horizontal, se sigue usando mucha codificación. Los campos principales del formato se resumen en la Tabla 17.6.

La ALU opera con las entradas que provienen de cuatro registros especializados no visibles por el usuario: A, B, C y D. El formato de microinstrucción contiene campos para cargar estos registros desde otros registros visibles por el usuario, realizar una función de la ALU y especificar un registro visible por el usuario para el almacenamiento del resultado. Hay también campos para carga y almacenamiento de datos entre registros y memoria.

El mecanismo de secuenciamiento del IBM 3033 se discutió en la Sección 17.2.

17.4 . TI 8800

La tarjeta de desarrollo de software (*Software Development Board*, SDB) Texas Instruments 8800 es una tarjeta que contiene un computador de 32 bits microprogramable. El sistema tiene una memoria de control que puede escribirse (*Writable Control Store*, WCS), implementada en RAM en lugar de en ROM. Este sistema no alcanza la velocidad o densidad de un sistema microprogramado con la memoria de control en ROM. Sin embargo, se usa para el desarrollo de prototipos y con fines educativos.

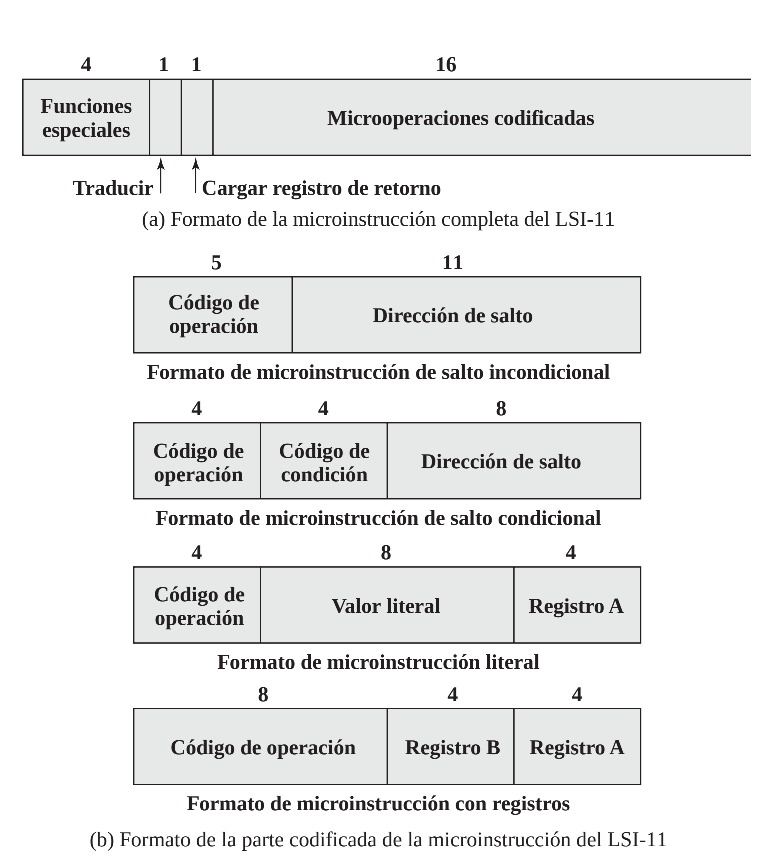


Figura 17.15. Formato de microinstrucción del LSI-11.

La SDB 8800 consta de los siguientes componentes (Figura 17.17):

- Memoria de microcódigo
- Microsecuenciador
- ALU de 32 bits
- Procesador de enteros y de coma flotante
- Memoria local de datos

Los componentes internos del sistema se conectan por medio de dos buses. El bus DA suministra el dato desde el campo de datos de la microinstrucción a la ALU, al procesador de coma flotante o al microsecuenciador. En el último caso, el dato consta de una dirección que se utiliza en una instrucción de salto. El bus se puede usar también para el suministro de datos desde la ALU o el microsecuenciador hacia otros componentes. El bus Y del sistema conecta la ALU y el procesador de coma flotante con la memoria local, y con los módulos externos por medio de la interfaz del PC.

La tarjeta se introduce en un computador anfitrión IBM PC o compatible. El computador anfitrión proporciona una plataforma adecuada para el ensamblaje y la depuración del microcódigo.

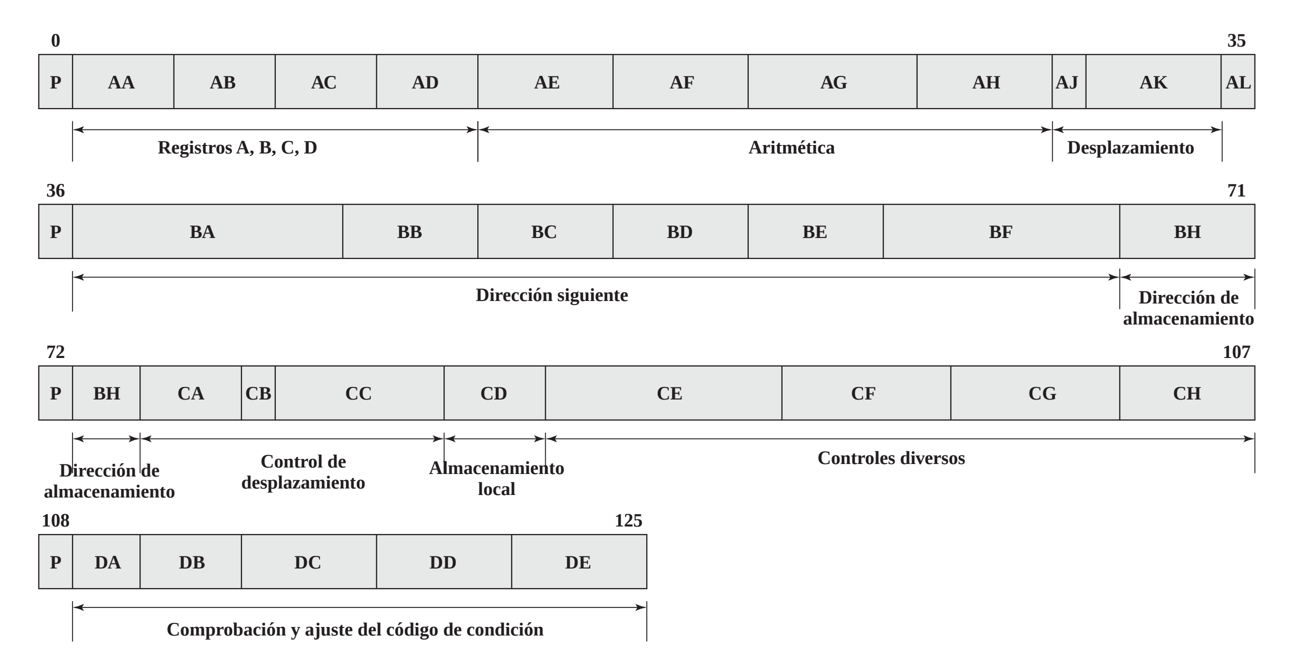


Figura 17.16. Formato de microinstrucción del IBM 3033.

Tabla 17.6. Campos de control de la microinstrucción del IBM 3033.

Campos de control de la ALU	
AA(3)	Carga registro A desde uno de los registros de datos
AB(3)	Carga registro B desde uno de los registros de datos
AC(3)	Carga registro C desde uno de los registros de datos
AD(3)	Carga registro D desde uno de los registros de datos
AE(4)	Encamina los bits especificados de A hacia la ALU
AF(4)	Encamina los bits especificados de B hacia la ALU
AG(5)	Especifica la operación aritmética de la ALU en la entrada A
AH(4)	Especifica la operación aritmética de la ALU en la entrada B
AJ(1)	Especifica la entrada D o B a la ALU en la parte B
AK(4)	Encamina la salida aritmética hacia el desplazador
CA(3)	Carga el registro F
CB(1)	Activa el desplazador
CC(5)	Especifica las funciones lógicas y de acarreo
CE(7)	Especifica el número de desplazamientos
Campos de secuenciamiento y de salto	
AL(1)	Finaliza la operación y realiza un salto
BA(8)	Fija los bits de orden superior (00-07) del registro de dirección de control
BB(4)	Especifica la condición para ajustar el bit 8 del registro de dirección de control
BC(4)	Especifica la condición para ajustar el bit 9 del registro de dirección de control
BD(4)	Especifica la condición para ajustar el bit 10 del registro de dirección de control
BE(4)	Especifica la condición para ajustar el bit 11 del registro de dirección de control
BF(4)	Especifica la condición para ajustar el bit 12 del registro de dirección de control

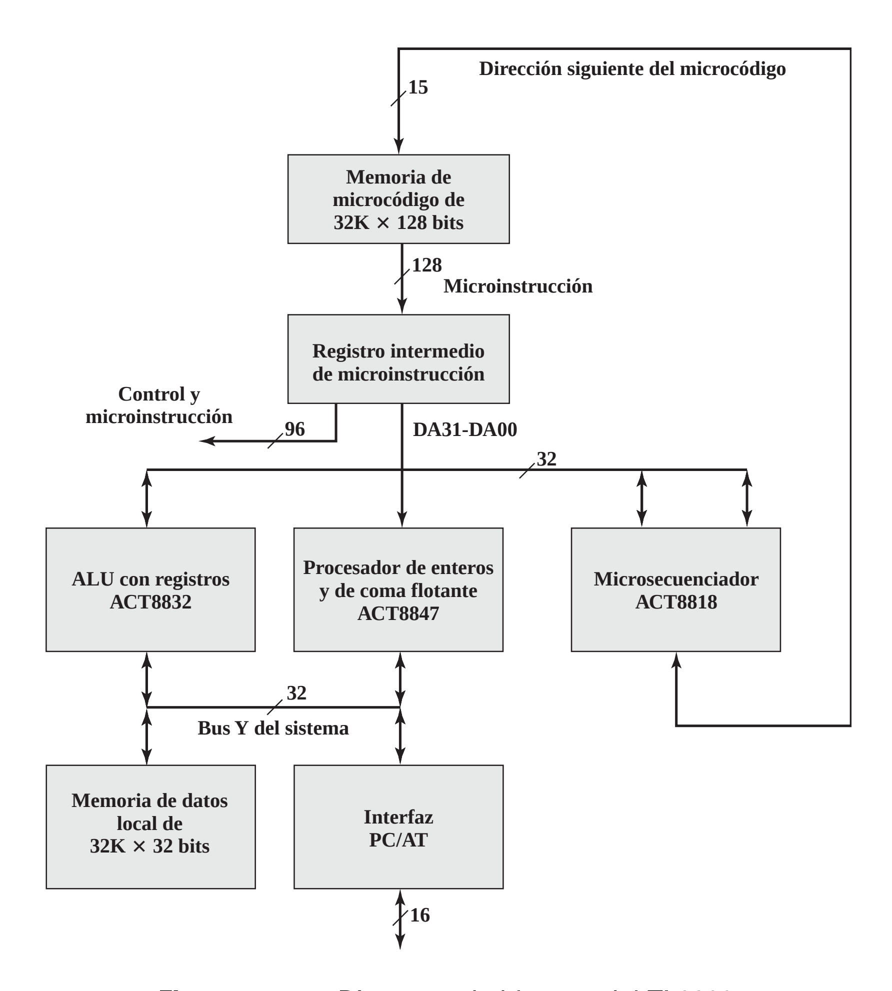


Figura 17.17. Diagrama de bloques del TI 8800.

FORMATO DE MICROINSTRUCCIÓN

El formato de microinstrucción del 8800 consta de 128 bits divididos en treinta campos funcionales, como se indica en la Tabla 17.7. Cada campo consta de uno o más bits y los campos se agrupan en cinco categorías principales:

- Control de la tarjeta
- Circuito procesador de enteros y de coma flotante 8847
- ALU con registros 8832
- Microsecuenciador 8818
- Campo de datos del WCS

Como se indica en la Figura 17.17, los 32 bits del campo de datos del WCS se introducen en el bus DA para suministrarlos como dato a la ALU, al procesador de coma flotante o al microsecuenciador.

Tabla 17.7. Formato de microinstrucciones del TI 8800.

Número de campo	Número de bits	Descripción
Control de la tarjeta		
1	5	Seleccionar entrada de código de condición
2	1	Habilitar/inhabilitar señal de petición de E/S externa
3	2	Habilitar/inhabilitar operaciones de lectura/escritura en la memoria de datos local
4	1	Cargar estado/no cargar estado
5	2	Determinar la unidad que controla el bus Y
6	2	Determinar la unidad que controla el bus DA
Circuito de procesamiento de enteros y de coma flotante 8847		
7	1	Control del registro C: señal de reloj activa o no
8	1	Seleccionar los bits más o menos significativos para el bus Y
9	1	Fuente de datos del registro C: ALU, multiplexor
10	4	Seleccionar modo IEEE o FAST para la ALU y el MUL
11	8	Seleccionar fuentes para operandos: registros RA, registros RB, registro P, registro S y registro C
12	1	Control del registro RB: señal de reloj activa o no
13	1	Control del registro RA: señal de reloj activa o no
14	2	Confirmación de la fuente de datos
15	2	Habilitar/inhabilitar registros intermedios
16	11	Función de la ALU del 8847
ALU con registros 8832		
17	2	Habilitar/no habilitar la salida de datos hacia el registro seleccionado: mitad más significativa, mitad menos significativa
18	2	Seleccionar la fuente de datos del banco de registros: bus DA, bus DB, salida ALU Y MUX, bus Y del sistema
19	3	Modificador de la instrucción de desplazamiento
20	1	Acarreo de entrada: ponerlo a 1 o a 0
21	2	Fijar el modo de configuración de la ALU: 32, 16 u 8 bits
22	2	Seleccionar entrada del multiplexor S: banco de registros, bus DB, registro MQ
23	1	Seleccionar entrada del multiplexor R: banco de registros, bus DA
24	6	Seleccionar registro del banco C para escritura
25	6	Seleccionar registro del banco B para lectura
26	6	Seleccionar registro del banco A para escritura
27	8	Función de la ALU
Microsecuenciador 8818		
28	12	Señales de control que entran al 8818
Campo de datos del WCS		
29	16	Bits más significativos del campo de datos del WCS
30	16	Bits menos significativos del campo de datos del WCS

Los otros 96 bits (campos 1-28) de la microinstrucción son señales de control que se introducen directamente en el módulo apropiado. Por simplicidad, estas conexiones no se muestran en la Figura 17.17.

Los seis primeros campos se ocupan de operaciones que están relacionadas con el control de la tarjeta, en lugar de controlar un componente particular. Las operaciones de control incluyen lo siguiente:

- Selección de códigos de condición para el control del secuenciador. El primer bit del campo 1 indica si el indicador de condición se va a poner a 1 o a 0, y los cuatro bits restantes indican cuál es ese indicador de condición.
- Envío de una petición de E/S al PC/AT.
- Habilitación de operaciones de lectura/escritura en la memoria local de datos.
- Determinación de la unidad que controla el bus Y del sistema. Se selecciona uno de los cuatro dispositivos que están conectados al bus (Figura 17.17).

Los últimos 32 bits son el campo de datos, que contiene información específica de cada microinstrucción particular.

Los campos restantes de la microinstrucción se estudian mejor en el contexto del dispositivo que controlan. En el resto de esta sección, examinaremos el microsecuenciador y la ALU con registros. Como la unidad de coma flotante no introduce nuevos conceptos, se omite.

MICROSECUENCIADOR

La función principal del microsecuenciador 8818 es generar la dirección de la siguiente microinstrucción del microprograma. Esta dirección de quince bits se suministra a la memoria de microcódigo (Figura 17.17).

La dirección siguiente se puede seleccionar desde una de las cinco fuentes:

1. El registro contador de microprograma (*Microprogram Counter Register*, MPC), usado para repetir (reutilizar la misma dirección) y continuar (incrementar la dirección en 1) la ejecución de microinstrucciones.
2. La pila, que da soporte a llamadas a subrutinas del microprograma así como a bucles iterativos y a retornos de interrupciones.
3. Los puertos DRA y DRB, que proporcionan dos caminos adicionales desde el hardware externo por los que se pueden enviar direcciones del microprograma. Estos dos puertos están conectados a los 16 bits más significativos y menos significativos del bus DA, respectivamente. Esto permite al microsecuenciador obtener la dirección de la siguiente microinstrucción a partir del campo de datos del WCS de la microinstrucción en curso o de un resultado calculado por la ALU.
4. Registros contadores RCA y RCB, que se pueden usar para el almacenamiento adicional de direcciones.
5. Una entrada externa en el puerto bidireccional Y para admitir interrupciones externas.

La Figura 17.18 es un diagrama de bloques lógico del 8818. El dispositivo consta de los siguientes grupos funcionales principales:

- Un contador de microprograma (MPC) de 16 bits compuesto por un registro y un incrementador.
- Dos registros contadores, RCA y RCB, para llevar la cuenta de bucles e iteraciones, almacenar direcciones de salto o controlar dispositivos externos.
- Una pila de 65 palabras de 16 bits, que permite llamadas a subrutinas de microprograma e interrupciones.
- Un registro de retorno de interrupción y una habilitación de salida Y para el procesamiento de interrupciones en el ámbito de las microinstrucciones.
- Un multiplexor de salida Y mediante el cual la dirección siguiente se puede seleccionar desde MPC, RCA, RCB, los buses externos DRA y DRB o la pila.

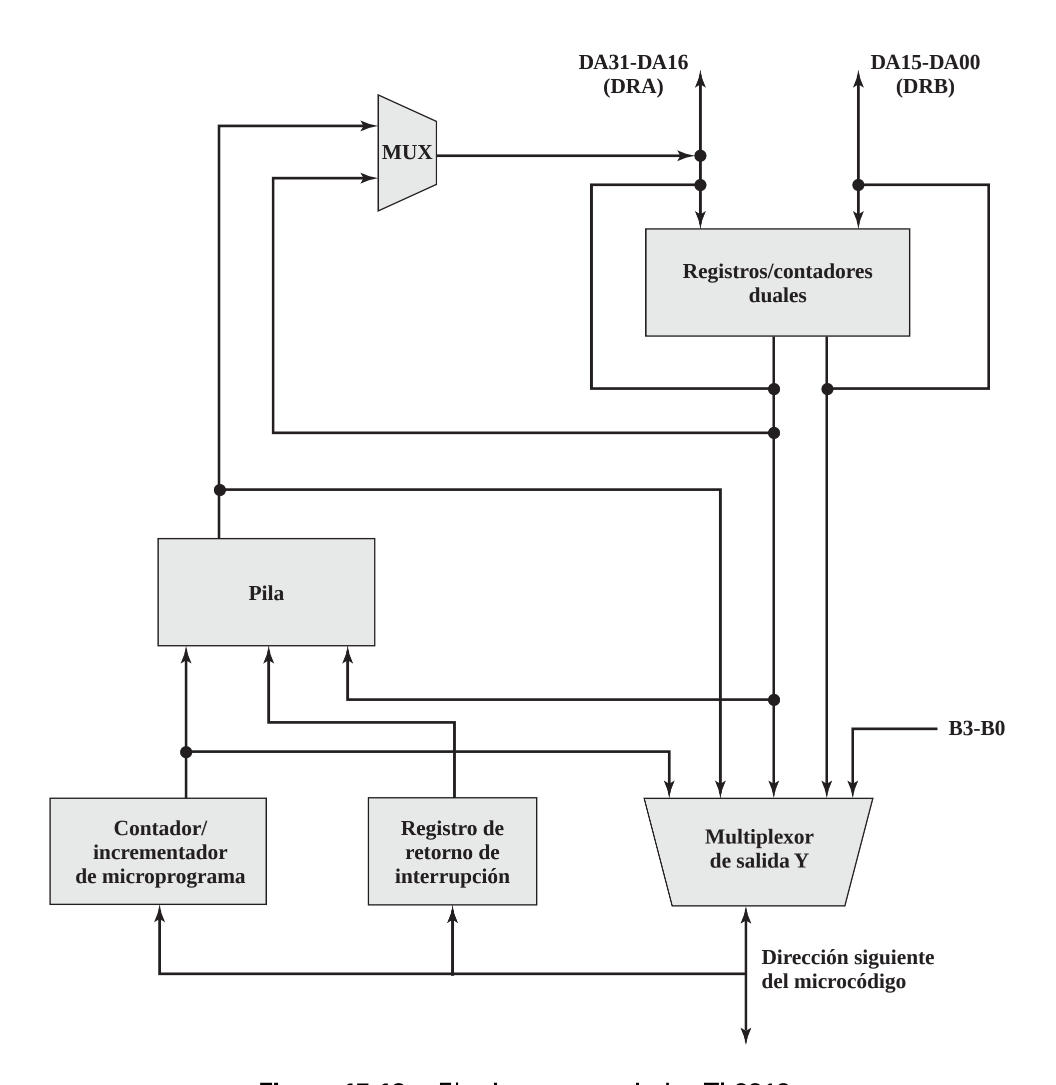


Figura 17.18. El microsecuenciador TI 8818.

Registros/Contadores. Los registros RCA y RCB se pueden cargar desde el bus DA, bien desde la microinstrucción en curso o bien desde la salida de la ALU. Los valores se pueden usar como contadores para controlar el flujo de ejecución y pueden decrementarse automáticamente cuando son accedidos. Los valores también se pueden utilizar como direcciones de microinstrucción y suministrarse al multiplexor de salida Y. Se dispone de un control independiente de ambos registros durante un mismo ciclo de microinstrucción con la excepción del decremento simultáneo de los dos registros.

Pila. La pila permite niveles múltiples de llamadas o interrupciones anidadas, y puede utilizarse tanto para saltos como para bucles. No hay que olvidar que estas operaciones se refieren a la unidad de control, no al procesador en su totalidad, y que las direcciones implicadas son las de microinstrucciones en la memoria de control.

Es posible realizar seis operaciones de pila:

1. Poner a cero, que pone el puntero de pila a cero, vaciando la pila.

2. Extraer, que decrementa el puntero de pila.

3. Apilar, que pone el contenido de MPC, del registro de retorno de interrupción, o del bus DRA en la pila e incrementa el puntero de pila.

4. Leer, que pone la dirección indicada por el puntero de lectura a disposición del multiplexor de salida Y.

5. Mantener, que hace que la dirección del puntero de pila permanezca inalterada.

6. Cargar el puntero de pila, que mueve los siete bits menos significativos del DRA al puntero de pila.

Control del microsecuenciador. El microsecuenciador está controlado principalmente por el campo de doce bits de la microinstrucción en curso, campo 28 (Tabla 17.7). Este campo consta de los siguientes subcampos:

- **OSEL (1 bit):** selección de salida. Determina cuál es el valor que se situará en la salida del multiplexor que introduce información en el bus DRA (esquina superior izquierda de la Figura 17.18). La salida se selecciona para que provenga de la pila o del registro RCA. DRA sirve como entrada al multiplexor de salida Y o al registro RCA.
- **SELDR (1 bit):** selección del bus DR. Si está puesto a 1, este bit selecciona el bus externo DA como entrada a los buses DRA/DRB. Si está puesto a 0, selecciona la salida del multiplexor DRA al bus DRA (controlado por OSEL) y el contenido de RBC al bus DRB.
- **ZEROIN (1 bit):** se utiliza para indicar un salto condicional. El comportamiento del microsecuenciador dependerá entonces del código de condición seleccionado en el campo 1 (Tabla 17.7).
- **RC2–RC0 (3 bits):** controles de registros. Estos bits determinan el cambio en los contenidos de los registros RCA y RCB. Cada registro puede mantenerse igual, decrementarse, o cargarse desde los buses DRA/DRB.
- **S2–S0 (3 bits):** controles de la pila. Estos bits determinan la operación que se va a realizar en la pila.

- **MUX2–MUX0:** controles de salida. Estos bits, junto con el código de condición si se usa, controlan el multiplexor de salida Y y por tanto la dirección de la próxima microinstrucción. El multiplexor puede seleccionar su salida desde la pila, DRA, DRB, o MPC.

El programador puede ajustar estos bits individualmente. Sin embargo, esto no es lo que se hace normalmente. En su lugar, el programador utiliza mnemotécnicos que equivalen a los patrones de bits que se requerirían normalmente. La Tabla 17.8 contiene una lista de los quince mnemotécnicos para el campo 28. Un ensamblador de microcódigo los convierte en los patrones de bits adecuados.

Por ejemplo, si el código de condición seleccionado actualmente es uno, la instrucción INC88181 se usa para seleccionar la siguiente microinstrucción secuencial. A partir de la Tabla 17.8, tenemos

$$\text{INC88181} = 000000111110$$

que se decodifica directamente como

- **OSEL = 0:** selecciona RCA como salida del MUX de salida a DRA; en este caso la selección es irrelevante.
- **SELDR = 0:** como se definió anteriormente; de nuevo, es irrelevante para esta instrucción.
- **ZEROIN = 0:** si se combina con el valor para MUX, indica que no se debe realizar el salto.
- **RC = 000:** conserva el valor actual de RA y RC.
- **S = 111:** conserva el estado actual de la pila.

Tabla 17.8. Bits de microinstrucción correspondientes al microsecuenciador TI 8818 (campo 28).

Mnemotécnico	Valor	Descripción
RST8818	000000000110	Reinicio de instrucción
BRA88181	011000111000	Saltar a la instrucción en DRA
BRA88180	010000111110	Saltar a la instrucción en DRA
INC88181	000000111110	Continuar instrucción
INC88180	001000001000	Continuar instrucción
CAL88181	010000110000	Salto a la subrutina en la dirección especificada por DRA
CAL88180	010000101110	Salto a la subrutina en la dirección especificada por DRA
RET8818	000000011010	Retornar de la subrutina
PUSH8818	000000110111	Apilar la dirección de retorno de interrupción en la pila
POP8818	100000010000	Retornar de interrupción
LOADDRA	000010111110	Cargar el contador DRA desde el bus DA
LOADDRB	000110111110	Cargar el contador DRB desde el bus DA
LOADDRAB	000110111100	Cargar DRA/DRB
DECRDRA	010001111100	Decrementar el contador DRA y saltar si no es cero
DECRDRB	010101111100	Decrementar el contador DRB y saltar si no es cero

- **MUX = 110:** elige MPC cuando el código de condición es 1 y DRA cuando el código de condición es 0.

ALU CON REGISTROS

El 8832 es una ALU de 32 bits con 64 registros que se puede configurar para funcionar como cuatro ALU de ocho bits, dos ALU de 16 bits o una única ALU de 32 bits.

El 8832 se controla con los 39 bits que componen los campos 17 a 27 de las microinstrucciones (Tabla 17.7); estos se suministran a la ALU como señales de control. Además, como se indica en la Figura 17.17, el 8832 tiene conexiones externas con el bus DA, de 32 bits y con el bus Y del sistema, también de 32 bits. Las entradas desde DA se pueden suministrar simultáneamente como datos de entrada al banco de registros de 64 palabras y al módulo de lógica de la ALU. Se proporciona una entrada desde el bus Y del sistema al módulo de lógica de la ALU. Los resultados de la ALU y de las operaciones de desplazamiento se llevan al bus DA o al bus Y del sistema. Estos resultados también pueden realimentar al banco de registros interno.

En el banco de registros hay tres puertos de direcciones de seis bits que permiten realizar simultáneamente una operación de captación de dos operandos y una escritura de un operando. El desplazador MQ y el registro MQ también pueden configurarse para funcionar independientemente e implementar operaciones de desplazamiento de doble precisión de ocho bits, 16 bits y 32 bits.

Los campos 17 a 26 de cada microinstrucción controlan el modo en el que los datos fluyen dentro del 8832 y entre el 8832 y el entorno externo. Los campos son los siguientes:

17. **Habilitación de escritura.** Estos dos bits especifican escritura de los 32 bits, de los 16 bits más significativos, de los 16 menos significativos o no escritura en el banco de registros. El registro destino se define en el campo 24.
18. **Selección de la fuente de datos del banco de registros.** Si va a ocurrir una escritura en el banco de registros, estos dos bits especifican la fuente: bus DA, bus DB, salida de la ALU o bus Y del sistema.
19. **Modificador de la instrucción de desplazamiento.** Especifica opciones relacionadas con el suministro de bits de relleno finales y con la lectura de los bits que se desplazan en las instrucciones de desplazamiento.
20. **Acarreo de entrada.** Este bit indica si se transmite un bit de acarreo a la ALU en la presente operación.
21. **Configuración del modo de la ALU.** El 8832 se puede configurar para operar como una única ALU de 32 bits, dos ALU de 16 bits o cuatro ALU de ocho bits.
22. **Entrada S.** Dos multiplexores internos, denominados multiplexores S y R, proporcionan las entradas del módulo de lógica de la ALU. Este campo selecciona la entrada que proporciona el multiplexor S: banco de registros, bus DB o registro MQ. El registro fuente está definido por el campo 25.
23. **Entrada R.** Selecciona la entrada que suministra el multiplexor R: banco de registros o bus DA.

24. **Registro destino.** Dirección en el banco de registros del registro a usar como operando destino.

25. **Registro fuente.** Dirección en el banco de registros del registro a usar como operando fuente, proporcionada por el multiplexor S.

26. **Registro fuente.** Dirección en el banco de registros del registro a usar como operando fuente, proporcionada por el multiplexor R.

Por último, el campo 27 es un código de operación de ocho bits que especifica la función aritmética o lógica que va a realizar la ALU. La Tabla 17.9 enumera las diferentes operaciones que se pueden llevar a cabo.

Como ejemplo de la codificación que se usa para especificar los campos 17 a 27, consideremos la instrucción que suma los contenidos del registro 1 y el registro 2, y deposita el resultado en el registro 3. La instrucción simbólica es

CONT11 [17], WELH, SELRFYMX, [24], R3, R2, R1, PASS + ADD

El ensamblador la traducirá al patrón de bits conveniente. Los componentes individuales de la instrucción se pueden describir como sigue:

- CONT11 es la instrucción básica NOP.
- El campo [17] se transforma en WELH (habilitación de escritura, baja y alta, *Write Enable, Low and High*) de manera que se escribe en un registro de 32 bits.
- El campo [18] se transforma en SELRFYMX para seleccionar la realimentación desde la salida ALU Y MUX.
- El campo [24] se transforma para designar el registro R3 como registro destino.
- El campo [25] se modifica para designar el registro R2 como uno de los registros fuente.
- El campo [26] se modifica para designar el registro R1 como uno de los registros fuente.
- El campo [27] se modifica para especificar una operación ALU de suma (ADD). La instrucción de desplazamiento de la ALU es PASS; por tanto, la salida de la ALU no se desplaza.

Se pueden destacar varios puntos sobre la notación simbólica. No es necesario especificar el número de campo para campos consecutivos. Es decir,

CONT11 [17], WELH, [18], SELRFYMX

se puede escribir como

CONT11 [17], WELH, SELRFYMX

ya que SELRFYMX está en el campo 18.

Las instrucciones de la ALU del Grupo 1 de la Tabla 17.9 se deben utilizar siempre conjuntamente con las del Grupo 2. Las de los Grupos 3-5 no deben usarse con las del Grupo 2.

Tabla 17.9. Campo de instrucciones de la ALU con registros del TI 8800 (campo 27).

Grupo 1		Función
ADD	H#01	R + S + Cn
SUBR	H#02	(NOT R) + S + Cn
SUBS	H#03	R + (NOT S) + Cn
INCS	H#04	S + Cn
INCNS	H#05	(NOT S) + Cn
INCR	H#06	R + Cn
INCNR	H#07	(NOT R) + Cn
XOR	H#09	R XOR S
AND	H#0A	R AND S
OR	H#0B	R OR S
NAND	H#0C	R NAND S
NOR	H#0D	R NOR S
ANDNR	H#0E	(NOT R) AND S
Grupo 2		**Función**
SRA	H#00	Desplazamiento aritmético a la derecha en precisión sencilla
SRAD	H#10	Desplazamiento aritmético a la derecha en precisión doble
SRL	H#20	Desplazamiento lógico a la derecha en precisión sencilla
SRLD	H#30	Desplazamiento lógico a la derecha en precisión doble
SLA	H#40	Desplazamiento aritmético a la izquierda en precisión sencilla
SLAD	H#50	Desplazamiento aritmético a la izquierda en precisión doble
SLC	H#60	Desplazamiento circular a la izquierda en precisión sencilla
SLCD	H#70	Desplazamiento circular a la izquierda en precisión doble
SRC	H#80	Desplazamiento circular a la derecha en precisión sencilla
SRCD	H#90	Desplazamiento circular a la derecha en precisión doble
MQSRA	H#A0	Desplazamiento aritmético a la derecha del registro MQ
MQSRL	H#B0	Desplazamiento lógico a la derecha del registro MQ
MQSLL	H#C0	Desplazamiento lógico a la izquierda del registro MQ
MQSLC	H#D0	Desplazamiento circular a la izquierda del registro MQ
LOADMQ	H#E0	Carga del registro MQ
PASS	H#F0	Pasar ALU a Y (sin desplazar)
Grupo 3		**Función**
SET1	H#08	Fijar bit 1
SET0	H#18	Fijar bit 0
TB1	H#28	Comprobar bit 1
TB0	H#38	Comprobar bit 0
ABS	H#48	Valor absoluto

(Continúa)

Tabla 17.9. Campo de instrucciones de la ALU con registros del TI 8800 (campo 27) *(continuación).*

SMTC	H#58	Signo y magnitud/complemento a dos
ADDI	H#68	Suma inmediata
SUBI	H#78	Resta inmediata
BADD	H#88	Suma (byte) R a S
BSUBS	H#98	Resta (byte) S de R
BSUBR	H#A8	Resta (byte) R de S
BINCS	H#B8	Incremento (byte) S
BINCNS	H#C8	Incremento negativo (byte) S
BXOR	H#D8	XOR (byte) R y S
BAND	H#E8	AND (byte) R y S
BOR	H#F8	OR (byte) R y S
Grupo 4		**Función**
CRC	H#00	Acumular caracteres de redundancia cíclica
SEL	H#10	Seleccionar S o R
SNORM	H#20	Normalización de longitud sencilla
DNORM	H#30	Normalización de longitud doble
DIVRF	H#40	Determinar resto de división
SDIVQF	H#50	Determinar cociente de división con signo
SMULI	H#60	Iterar multiplicación con signo
SMULT	H#70	Terminar multiplicación con signo
SDIVIN	H#80	Inicializar división con signo
SDIVIS	H#90	Comenzar división con signo
SDIVI	H#A0	Iterar división con signo
UDIVIS	H#B0	Comenzar división sin signo
UDIVI	H#C0	Iterar división sin signo
UMULI	H#D0	Iterar multiplicación sin signo
SDIVIT	H#E0	Terminar división con signo
UDIVIT	H#F0	Terminar división sin signo
Grupo 5		**Función**
LOADFF	H#0F	Cargar los biestables de división/BCD
CLR	H#1F	Borrar
DUMPFF	H#5F	Enviar a la salida los biestables de división/BCD
BCDBIN	H#7F	BCD a binario
EX3BC	H#8F	Corrección de byte en exceso −3
EX3C	H#9F	Corrección de palabra en exceso −3
SDIVO	H#AF	Comprobación de desbordamiento en la división con signo
BINEX3	H#DF	Binario a exceso −3
NOP32	H#FF	No operación

17.5. LECTURAS RECOMENDADAS

Hay varios libros dedicados a la microprogramación. Quizás el más completo es [LYNC93]. [SEGE91] presenta los fundamentos de la microcodificación y el diseño de sistemas microcodificados, mediante un diseño paso a paso de un procesador sencillo de 16 bits. [CART96] también presenta los conceptos básicos usando una máquina de ejemplo. [PARK89] y [TI90] proporcionan una descripción detallada de la tarjeta de desarrollo de software TI 8800.

[VASS03] discute la evolución del uso de microcódigo en el diseño de computadores y su estado actual.

CART96 Carter, J.: *Microprocessor Architecture and Microprogramming.* Upper Saddle River, NJ. Prentice Hall, 1996.

LYNC93 Lynch, M.: *Microprogrammed State Machine Design.* Boca Raton, FL. CRC Press, 1993.

PARK89 Parker, A. y Hamblen, J.: *An Introduction to Microprogramming with Exercises Designed for the Texas Instruments SN74ACT8800 Software Development Board.* Dallas, TX. Texas Instruments, 1989.

SEGE91 Segee, B. y Field, J.: *Microprogramming and Computer Architecture.* New York. Wiley, 1991.

TI90 Texas Instruments Inc. *SN74ACT8800 Family Data Manual.* SCSS006C, 1990.

VASS03 Vassiliadis, S.; Wong, S. y Cotofana, S.: «Microcode Processing: Positioning and Directions». *IEEE Micro,* julio-agosto, 2003.

17.6. PALABRAS CLAVE, PREGUNTAS DE REPASO Y PROBLEMAS

PALABRAS CLAVE

codificación de microinstrucciones
ejecución de microinstrucciones
firmware
lenguaje de microprogramación
memoria de control
microinstrucción desempaquetada
microinstrucción horizontal
microinstrucción vertical
microinstrucciones
microprograma
microprogramación *hard*
microprogramación *soft*
palabra de control
secuenciamiento de microinstrucciones
unidad de control microprogramada

PREGUNTAS DE REPASO

17.1. ¿Cuál es la diferencia entre una implementación cableada y una implementación microprogramada de una unidad de control?

17.2. ¿Cómo se interpreta una microinstrucción horizontal?

17.3. ¿Cuál es el propósito de la memoria de control?

17.4. ¿Cuál es la secuencia típica de ejecución de una microinstrucción horizontal?

17.5. ¿Cuál es la diferencia entre las microinstrucciones horizontales y verticales?

17.6. ¿Cuáles son las tareas básicas realizadas por una unidad de control microprogramada?

17.7. ¿Cuál es la diferencia entre las microinstrucciones empaquetadas y desempaquetadas?

17.8. ¿Cuál es la diferencia entre la microprogramación *hard* y *soft*?

17.9. ¿Cuál es la diferencia entre la codificación funcional y la codificación por recursos?

PROBLEMAS

17.1. Describa la implementación de la instrucción de multiplicación en la máquina hipotética diseñada por Wilkes. Utilice una descripción narrada y un organigrama.

17.2. Suponga un repertorio de microinstrucciones que incluye una microinstrucción con la siguiente forma simbólica:

$$\text{IF } (AC_0 = 1) \text{ THEN CAR} \leftarrow (C_{0-6}) \text{ ELSE CAR} \leftarrow (\text{CAR}) + 1$$

donde AC_0 es el bit de signo del acumulador y C_{0-6} son los siete primeros bits de la microinstrucción. Utilizando esta microinstrucción, escriba un microprograma que implemente una instrucción máquina *Branch Register Minus* (BRM), que salte si el AC es negativo. Suponga que los bits C_1 a C_n de la microinstrucción especifican un conjunto paralelo de microoperaciones. Exprese el programa simbólicamente.

17.3. Un procesador sencillo tiene cuatro fases principales en su ciclo de instrucción: captación, ciclo indirecto, ejecución e interrupción. Dos indicadores de un bit señalan la fase en curso en una implementación cableada.

(a) ¿Por qué se necesitan estos indicadores?

(b) ¿Por qué no se necesitan en una unidad de control microprogramada?

17.4. Considere la unidad de control de la Figura 17.7. Suponga que la memoria de control tiene una anchura de 24 bits. La parte de control del formato de microinstrucción está dividida en dos campos. Un campo de microoperación de trece bits especifica las microoperaciones que se van a realizar. Un campo de selección de dirección especifica una condición, basada en los indicadores, que originará un salto de microinstrucción. Hay ocho indicadores.

(a) ¿Cuántos bits hay en el campo de selección de dirección?

(b) ¿Cuántos bits hay en el campo de dirección?

(c) ¿Cuál es el tamaño de la memoria de control?

17.5. ¿Cómo se puede realizar un salto incondicional bajo las circunstancias del problema anterior? ¿Cómo se puede evitar el salto?; es decir, describa una microinstrucción que no especifique ningún salto, ni condicional ni incondicional.

17.6. Deseamos proporcionar ocho palabras de control para cada rutina de instrucción máquina. Los códigos de operación de la instrucción máquina tienen cinco bits, y la memoria de control tiene 1 024 palabras. Sugiera una traducción del registro de instrucción al registro de dirección de control.

17.7. Se va a usar un formato de microinstrucción codificado. Muestre cómo un campo de microoperación de nueve bits puede dividirse en subcampos para especificar 46 acciones diferentes.

17.8. Un procesador tiene 16 registros, una ALU con 16 funciones lógicas y 16 funciones aritméticas, y un desplazador con ocho operaciones, todo conectado por un bus interno del procesador. Diseñe el formato de microinstrucción para especificar las distintas microoperaciones del procesador.

PARTE 5

ORGANIZACIÓN PARALELA

CUESTIONES A TRATAR EN LA QUINTA PARTE

La parte final del libro considera los aspectos de la cada vez más importante organización paralela. En una organización paralela, varias unidades de proceso cooperan en la ejecución de aplicaciones. Mientras que un procesador superescalar aprovecha las posibilidades de ejecución paralela en el nivel de instrucciones, una organización de procesamiento paralelo utiliza un nivel de paralelismo más grueso que permite que varios procesadores puedan ejecutar el trabajo a realizar en paralelo, y cooperativamente. Existen una serie de cuestiones que deben resolverse en este tipo de organizaciones. Por ejemplo, si muchos procesadores, cada uno con su caché, comparten el acceso a la misma memoria, deben emplearse mecanismos hardware o software para asegurar que ambos procesadores comparten una imagen válida de la memoria principal. Esto se conoce como problema de coherencia de caché. Este aspecto, junto con otros, se analizará en la Parte Cinco.

ESQUEMA DE LA PARTE V

CAPÍTULO 18. PROCESAMIENTO PARALELO

El Capítulo 18 proporciona una revisión de las cuestiones relativas al procesamiento paralelo. Después, el capítulo considera tres organizaciones de varios procesadores: los multiprocesadores simétricos (SMP, *symmetric multiprocessors*), los *clusters*, y las máquinas de acceso a memoria no uniforme (NUMA, *nonuniform memory access*). Los SMP y los *clusters* son las formas más comunes de conseguir mejorar las prestaciones y la disponibilidad mediante el uso de varios procesadores. Los sistemas NUMA constituyen un concepto más reciente que todavía no ha alcanzado un éxito comercial amplio, aunque representan una alternativa muy prometedora. Por último, el Capítulo 18 considera una organización de propósito específico conocida como procesador vectorial.

CAPÍTULO 18

Procesamiento paralelo

PUNTOS CLAVE

- Una manera tradicional de incrementar las prestaciones de un sistema consiste en utilizar varios procesadores que puedan ejecutar en paralelo una carga de trabajo dada. Las dos organizaciones de múltiples procesadores más comunes son los **multiprocesadores simétricos** (SMP) y los *clusters*. Recientemente, los sistemas de **acceso no uniforme a memoria** (NUMA) han aparecido comercialmente.
- Un SMP es un computador constituido por varios procesadores similares, interconectados mediante un bus o algún tipo de estructura de conmutación. El problema más crítico a resolver en un SMP es la coherencia de caché. Cada procesador tiene su propia caché, y es posible que una línea de datos dada esté presente en más de una caché. Si esa línea se altera en una caché, entonces tanto la memoria principal como las otras cachés tienen versiones no válidas de dicha línea.
- Cuando en un único chip se implementan varios procesadores, se habla de **multiprocesador monochip**. Un diseño relacionado consiste en repetir algunos componentes de un procesador para que este pueda ejecutar varias hebras concurrentemente. Es lo que se conoce como **procesador multihebra**.
- Un ***cluster*** es un grupo de computadores completos interconectados y trabajando juntos como un solo recurso de cómputo, proporcionando la ilusión de ser una única máquina. El término *computador completo* significa que puede funcionar autónomamente, fuera del *cluster*.
- Un sistema NUMA es un multiprocesador de memoria compartida en el que el tiempo de acceso de un procesador a una palabra de memoria varía con la ubicación de la palabra en memoria.
- Los procesadores vectoriales son procesadores paralelos de propósito específico, diseñados para procesar eficientemente vectores o matrices de datos.

Tradicionalmente, el computador se ha visto como una máquina secuencial. La mayoría de los lenguajes de programación del computador requieren que el programador especifique los algoritmos mediante una secuencia de instrucciones. Los procesadores ejecutan los programas procesando instrucciones máquina de una en una. Cada instrucción se ejecuta mediante una secuencia de operaciones (captar instrucción, captar operandos, realizar la operación, almacenar los resultados).

Esta perspectiva del computador no es completamente cierta. En el nivel de microoperación, se generan al mismo tiempo múltiples señales de control. La segmentación de las instrucciones, al menos en cuanto al solapamiento de las operaciones de captación y ejecución, se ha utilizado desde hace tiempo. Ambos casos son ejemplos de funciones que se realizan en paralelo. Es el mismo enfoque de la organización superescalar, que aprovecha el paralelismo entre instrucciones. Un procesador superescalar dispone de varias unidades de ejecución que pueden ejecutar en paralelo varias instrucciones del mismo programa.

A medida que la tecnología de los computadores se ha desarrollado, y ha disminuido el costo del hardware del computador, los diseñadores de computadores han visto más y más posibilidades en el paralelismo, normalmente para mejorar las prestaciones y, en algunos casos, para mejorar la fiabilidad. Después de una revisión de conceptos, en este capítulo se examinan las tres organizaciones paralelas de más éxito. En primer lugar se estudian los multiprocesadores simétricos (SMP), una de las primeras y todavía el ejemplo más común de organización paralela. Un SMP incluye varios procesadores que comparten la memoria principal común. La organización SMP pone de manifiesto el problema de la coherencia de caché, al que se dedica una sección específica. A continuación, el capítulo aborda los procesadores multihebra y los multiprocesadores monochip. Después se describen los *clusters*, que están constituidos por varios computadores independientes organizados para poder trabajar cooperativamente. Los *clusters* han llegado a ser bastante comunes a la hora de procesar cargas de trabajo que sobrepasan la capacidad de un SMP. La tercera aproximación al uso de varios procesadores está representada por las máquinas de acceso no uniforme a memoria (NUMA). La alternativa NUMA es relativamente nueva y todavía no se ha extendido comercialmente, pero a menudo se considera la alternativa a los computadores SMP y a los *clusters*. Finalmente, el capítulo estudia las aproximaciones hardware a la computación vectorial. Estas propuestas optimizan la ALU para el procesamiento de vectores o matrices de números en coma flotante. Se han utilizado en la implementación de los sistemas conocidos como *supercomputadores*.

18.1. ORGANIZACIONES CON VARIOS PROCESADORES

TIPOS DE SISTEMAS DE PARALELOS

La taxonomía introducida primeramente por Flynn [FLYN72] es todavía la forma más común de clasificar a los sistemas según sus capacidades de procesamiento paralelo. Flynn propuso las siguientes categorías o clases de computadores:

- **Una secuencia de instrucciones y una secuencia de datos (SISD, *Single Instruction Single Data*)**: un único procesador interpreta una única secuencia de instrucciones para operar con los datos almacenados en una única memoria. Los computadores monoprocesador caen dentro de esta categoría.
- **Una secuencia de instrucciones y múltiples secuencias de datos (SIMD, de *Single Instruction Multiple Data*):** una única instrucción máquina controla paso a paso la ejecución simultánea y sincronizada de un cierto número de elementos de proceso. Cada elemento de proceso tiene una memoria asociada, de forma que cada instrucción es ejecutada por cada procesador con un conjunto de datos diferentes. Los procesadores vectoriales y los matriciales pertenecen a esta categoría.
- **Múltiples secuencias de instrucciones y una secuencia de datos (MISD):** se transmite una secuencia de datos a un conjunto de procesadores, cada uno de los cuales ejecuta una secuencia de instrucciones diferente. Esta estructura nunca ha sido implementada.
- **Múltiples secuencias de instrucciones y múltiples secuencias de datos (MIMD):** un conjunto de procesadores ejecuta simultáneamente secuencias de instrucciones diferentes con conjuntos de datos diferentes. Los SMP, los *clusters* y los sistemas NUMA son ejemplos de esta categoría.

En la organización MIMD los procesadores son de uso general; cada uno es capaz de procesar todas las instrucciones necesarias para realizar las transformaciones apropiadas de los datos. Los computadores MIMD se pueden subdividir además según la forma que tienen los procesadores para comunicarse (Figura 18.1). Si los procesadores comparten una memoria común, entonces cada procesador accede a los programas y datos almacenados en la memoria compartida, y los procesadores se comunican unos con otros a través de esa memoria. La forma más común de este tipo de sistemas se conoce como **multiprocesador simétrico (SMP)**, que se examinará en la Sección 18.2. En un SMP, varios procesadores comparten una única memoria mediante un bus compartido u otro tipo de mecanismo de interconexión. Una característica distintiva de estos sistemas es que el tiempo de acceso a memoria principal es aproximadamente el mismo para cualquier procesador. Un desarrollo más reciente es la organización con **acceso no uniforme a memoria (NUMA)**, que se describe en la Sección 18.5. Como el propio nombre indica, el tiempo de acceso a zonas de memoria diferentes puede diferir en un computador NUMA.

Un conjunto de computadores monoprocesador independientes o de SMP pueden interconectarse para formar un ***cluster***. La comunicación entre los computadores se realiza bien mediante conexiones fijas o mediante algún tipo de red.

ORGANIZACIONES PARALELAS

La Figura 18.2 muestra los esquemas generales de las clases de la taxonomía de la Figura 18.1. La Figura 18.2a corresponde a la estructura de un SISD. Se dispone de una unidad de control (UC) que proporciona una secuencia de instrucciones (SI) a una unidad de proceso (UP). La unidad de proceso actúa sobre una única secuencia de datos (SD) captados desde la unidad de memoria (UM). En una máquina SIMD, también existe una sola unidad de control, que proporciona una única

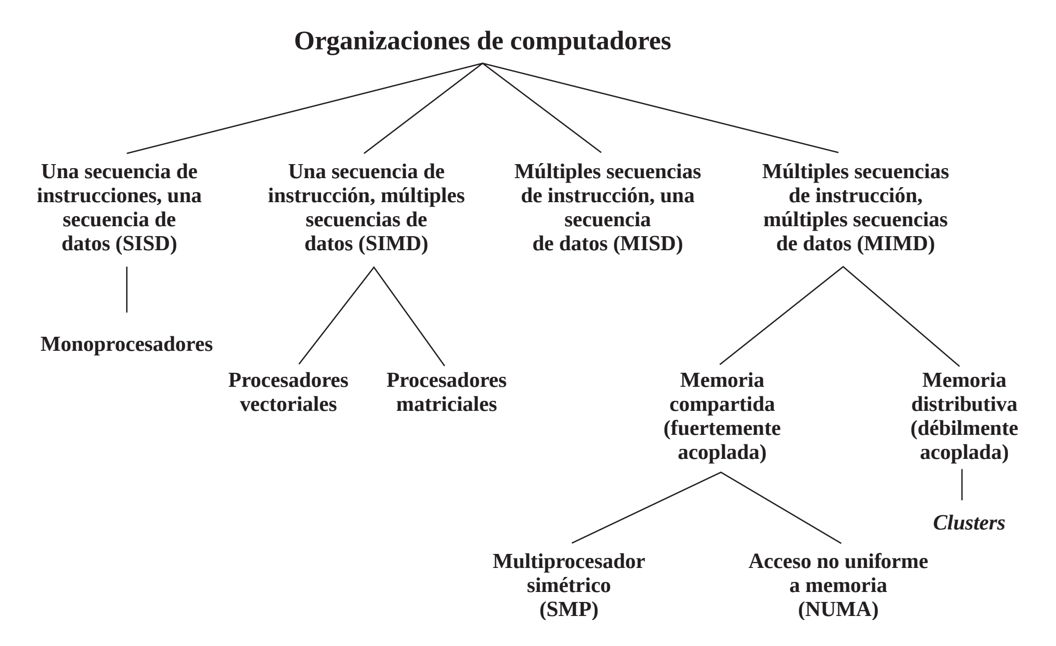


Figura 18.1. Taxonomía de las arquitecturas paralelas.

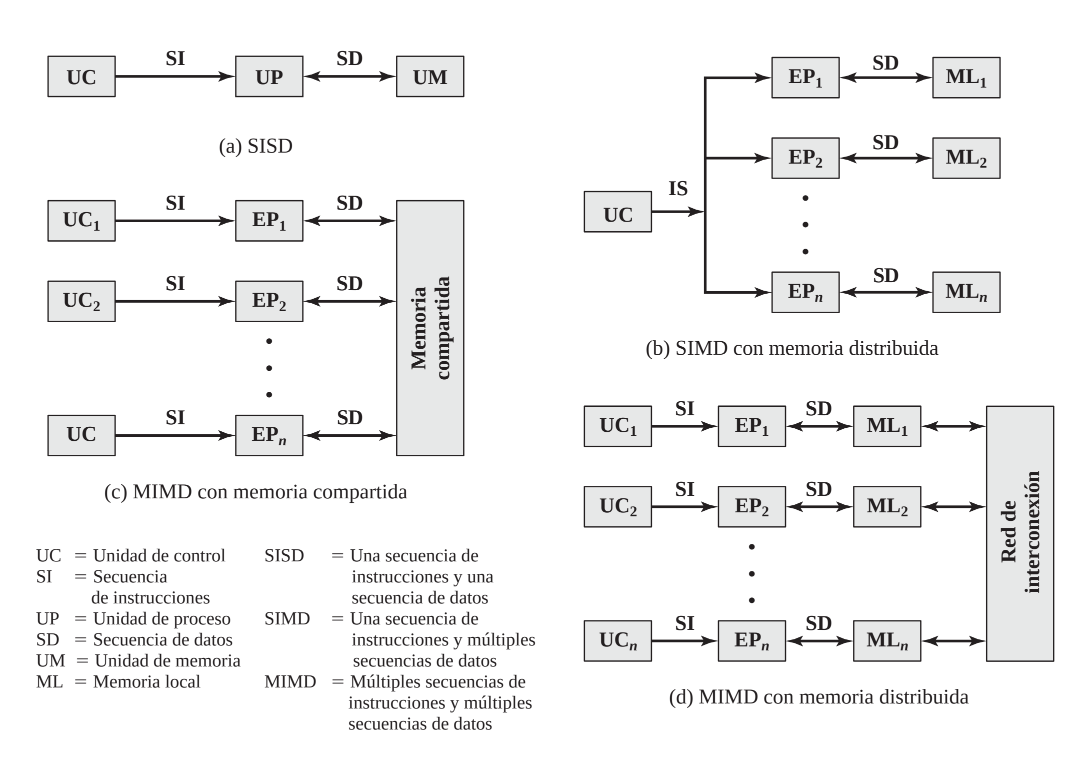


Figura 18.2. Organizaciones de computador alternativas.

secuencia de instrucciones a cada elemento de proceso. Cada elemento de proceso puede tener su propia memoria dedicada (mostrada en la Figura 18.2b) o puede ser una memoria compartida. Finalmente, en un computador MIMD hay múltiples unidades de control, y cada una proporciona una secuencia de instrucciones separada a su propio elemento de proceso. El MIMD puede ser un multiprocesador de memoria compartida (Figura 18.2c) o un multicomputador de memoria distribuida (Figura 18.2d).

Los aspectos de diseño relacionados con los SMP, los *clusters* y los NUMA son complejos, implicando cuestiones relativas a la organización física, las estructuras de interconexión, el diseño de los sistemas operativos, y el software de las aplicaciones. Nuestro interés se centra fundamentalemte en la organización, aunque se describirán brevemente aspectos del diseño de los sistemas operativos.

18.2. MULTIPROCESADORES SIMÉTRICOS

Hasta hace poco, prácticamente todos los computadores personales y estaciones de trabajo utilizaban un único microprocesador de uso general. A medida que aumenta la demanda de mayores prestaciones y dado que el coste de los microprocesadores continúa reduciéndose, los fabricantes han introducido los sistemas SMP. El término *SMP* se refiere a la arquitectura hardware del computador y también al comportamiento del sistema operativo que utiliza dicha arquitectura. Un SMP puede definirse como un computador con las siguientes características:

1. Hay dos o más procesadores similares de capacidades comparables.

2. Estos procesadores comparten la memoria principal y las E/S y están interconectados mediante un bus u otro tipo de sistema de interconexión, de forma que el tiempo de acceso a memoria es aproximadamente el mismo para todos los procesadores.

3. Todos los procesadores comparten los dispositivos de E/S, bien a través de los mismos canales o mediante canales distintos que proporcionan caminos de acceso al mismo dispositivo.

4. Todos los procesadores pueden desempeñar las mismas funciones (de ahí el término *simétrico*).

5. El sistema está controlado por un sistema operativo integrado que proporciona la interacción entre los procesadores y sus programas a los niveles de trabajo, tarea, fichero y datos.

El significado de los puntos 1 a 4 es claro. El punto 5 corresponde a una de las diferencias con los sistemas multiprocesador débilmente acoplados, tales como los *clusters*. En estos, la unidad de interacción física es normalmente un mensaje o un fichero completo. En un SMP, la interacción se puede producir a través de elementos de datos individuales, y puede existir un elevado nivel de cooperación entre procesadores.

El sistema operativo de un SMP planifica la distribución de procesos o hilos (*threads*) entre todos los procesadores. Un SMP tiene las siguientes ventajas potenciales con respecto a una arquitectura monoprocesador:

- **Prestaciones:** si el trabajo a realizar por un computador puede organizarse de forma que partes del mismo se puedan realizar en paralelo, entonces un sistema con varios procesadores proporcionará mejores prestaciones que uno con un solo procesador del mismo tipo (Figura 18.3).

- **Disponibilidad:** en un multiprocesador simétrico, debido a que todos los procesadores pueden realizar las mismas funciones, un fallo en un procesador no hará que el computador se detenga.

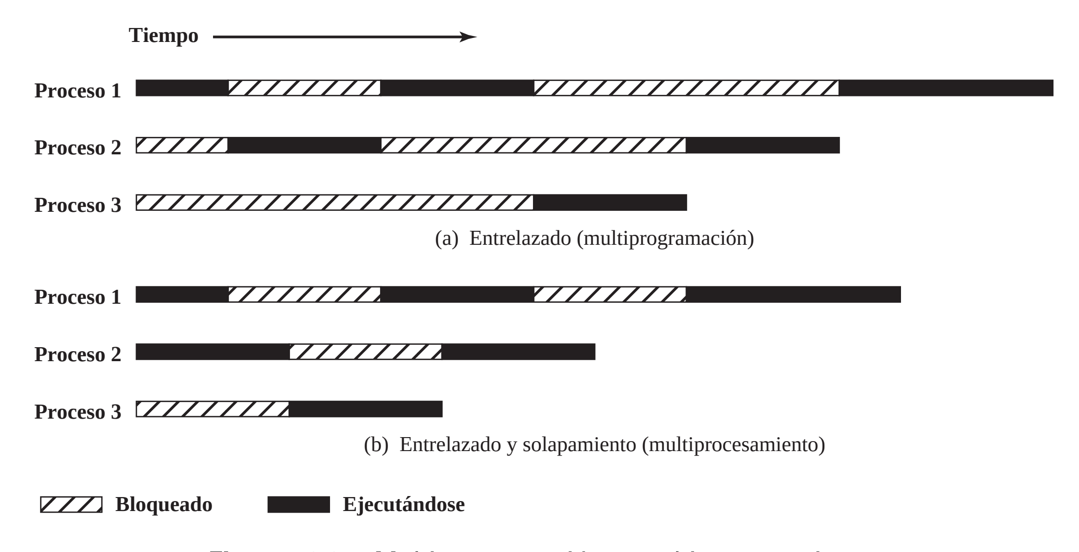


Figura 18.3. Multiprogramación y multiprocesamiento.

- **Crecimiento incremental:** se pueden aumentar las prestaciones del sistema añadiendo más procesadores.
- **Escalado:** los fabricantes pueden ofrecer una gama de productos con precios y prestaciones diferentes en función del número de procesadores que configuran el sistema.

Es importante resaltar que los anteriores son beneficios potenciales más que beneficios garantizados. El sistema operativo debe disponer de herramientas y funciones que permitan explotar el paralelismo presente en un SMP.

Una característica atractiva de un SMP es que la existencia de varios procesadores es transparente al usuario. El sistema operativo se encarga de la sincronización entre los procesadores y de la planificación de los hilos o de los procesos, asignándolos a los distintos procesadores.

ORGANIZACIÓN

La Figura 18.4 describe en términos generales la organización de un sistema multiprocesador. Hay dos o más procesadores. Cada procesador es autónomo, incluyendo una unidad de control, una ALU, registros y, posiblemente, caché. Cada procesador tiene acceso a una memoria principal compartida y a los dispositivos de E/S a través de alguna forma de mecanismo de interconexión. Los procesadores pueden comunicarse entre sí a través de la memoria (mensajes e información de control almacenada en áreas comunes para datos). También es posible que los procesadores intercambien señales directa-

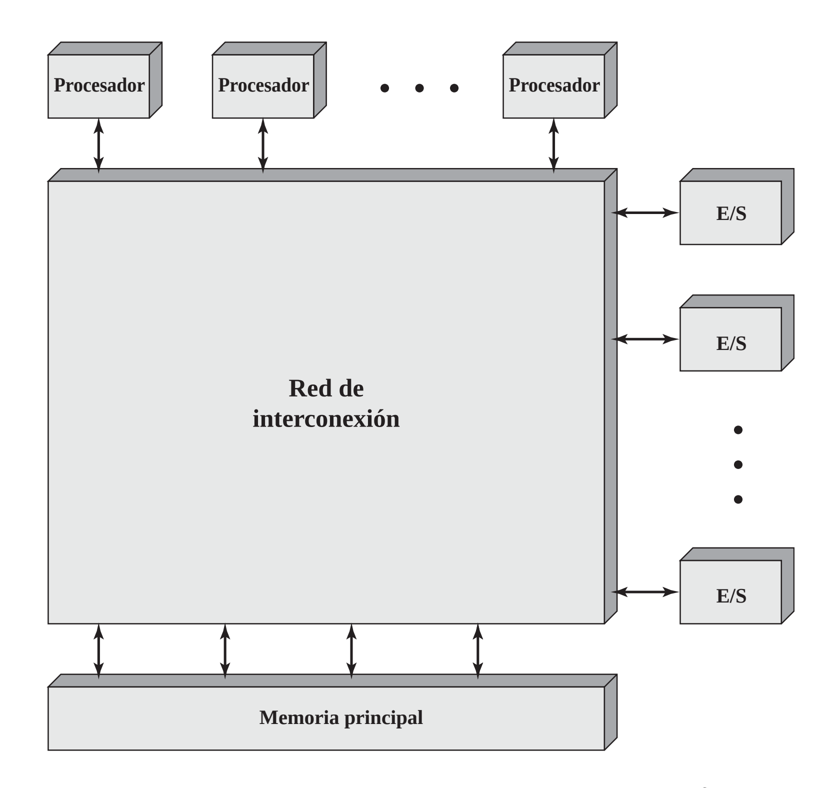


Figura 18.4. Diagrama de bloques genérico de un multiprocesador fuertemente acoplado.

mente. La memoria a menudo se organiza de forma que sean posibles los accesos simultáneos a bloques de memoria separados. En algunas configuraciones, cada procesador puede tener también su propia memoria principal privada y sus canales de E/S, además de los recursos compartidos.

BUS DE TIEMPO COMPARTIDO

La organización más común en los computadores personales, estaciones de trabajo y servidores es el bus de tiempo compartido. El bus de tiempo compartido es el mecanismo más simple para construir un sistema multiprocesador (Figura 18.5). La estructura y las interfaces son básicamente las mismas que las de un sistema de un único procesador que utilice un bus para la interconexión. El bus consta de líneas de control, dirección y datos. Para facilitar las transferencias de DMA con los procesadores de E/S, se proporcionan los elementos para el:

- **Direccionamiento:** debe ser posible distinguir los módulos del bus para determinar la fuente y el destino de los datos.
- **Arbitraje:** cualquier módulo de E/S puede funcionar temporalmente como un «maestro». Se proporciona un mecanismo para arbitrar entre las peticiones que compiten por el control del bus, utilizando algún tipo de esquema de prioridad.

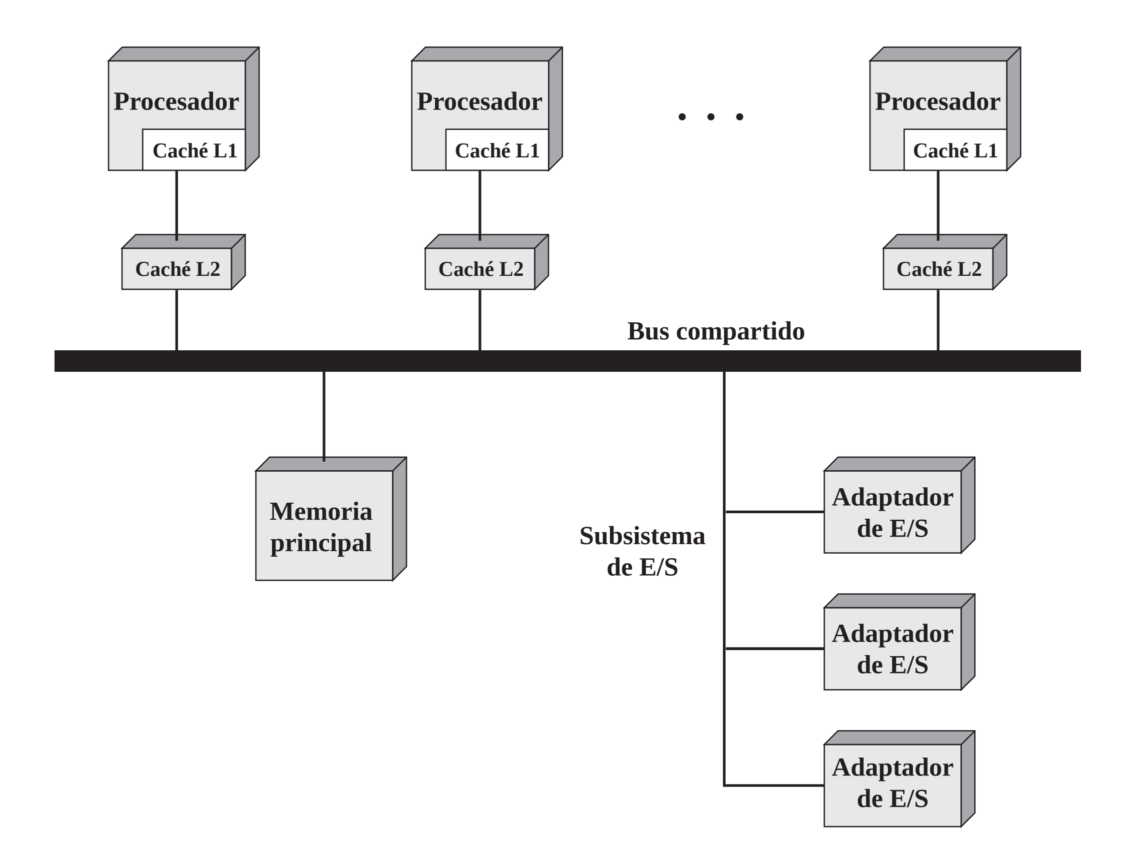


Figura 18.5. Organización de multiprocesador simétrico.

- **Tiempo Compartido:** cuando un módulo está controlando el bus, los otros módulos no tienen acceso al mismo y deben, si es necesario, suspender su operación hasta que dispongan del bus.

Estas características monoprocesador son utilizables directamente en una configuración de SMP. En este caso, hay varias CPU además de varios procesadores de E/S que intentan tener acceso a uno o más módulos de memoria a través del bus.

La organización del bus presenta varias características atractivas:

- **Simplicidad:** es la aproximación más simple para organizar el multiprocesador. La interfaz física y la lógica de cada procesador para el direccionamiento, el arbitraje y para compartir el tiempo del bus es el mismo que el de un sistema con un solo procesador.
- **Flexibilidad:** es generalmente sencillo expandir el sistema conectando más procesadores al bus.
- **Fiabilidad:** el bus es esencialmente un medio pasivo, y el fallo de cualquiera de los dispositivos conectados no provocaría el fallo de todo el sistema.

La principal desventaja de la organización de bus son las prestaciones. Todas las referencias a memoria pasan por el bus. En consecuencia, la velocidad del sistema está limitada por el tiempo de ciclo. Para mejorar las prestaciones, es deseable equipar a cada procesador de una memoria caché. Esta reduciría dramáticamente el número de accesos. Típicamente, los PC y las estaciones de trabajo de tipo SMP tienen dos niveles de caché, una caché L1 interna (en el mismo chip que el procesador) y una caché L2 externa o interna. Algunos procesadores actuales también utilizan una memoria caché L3.

El uso de cachés introduce algunas consideraciones de diseño nuevas. Puesto que cada caché local contiene una imagen de una parte de la memoria, si se altera una palabra en una caché,es concebible que eso podría invalidar una palabra en otra caché. Para evitarlo, se debe avisar a los otros procesadores de que se ha producido una actualización de memoria. Este problema se conoce como problema de *coherencia de caché*, que es resuelto típicamente por el hardware más que por el sistema operativo. La Sección 18.3 trata este punto.

CONSIDERACIONES DE DISEÑO DE UN SISTEMA OPERATIVO DE MULTIPROCESADOR

Un sistema operativo de SMP gestiona los procesadores y demás recursos del computador para que el usuario perciba un solo sistema operativo controlando los recursos del sistema. De hecho, el computador debería parecer un sistema monoprocesador con multiprogramación. Tanto en un SMP como en un sistema monoprocesador, pueden estar activos varios trabajos o procesos al mismo tiempo, y es responsabilidad del sistema operativo planificar su ejecución y asignar los recursos. Un usuario puede desarrollar aplicaciones que utilizan varios procesos o varios hilos dentro de un proceso sin tener en cuenta si se dispone de uno o de varios procesadores. Así, un sistema operativo de multiprocesador debe proporcionar toda la funcionalidad de un sistema operativo con multiprogramación más, las características adicionales que permitan utilizar varios procesadores. Entre los puntos clave de diseño están:

- **Procesos concurrentes simultáneos:** las rutinas del sistema operativo deben ser reentrantes para permitir que varios procesadores puedan ejecutar simultáneamente el mismo código IS.

Con varios procesadores ejecutando la misma o distintas partes del sistema operativo, las tablas y las estructuras de gestión del sistema operativo deben manejarse apropiadamente para evitar bloqueos u operaciones no válidas.

- **Planificación:** la planificación puede realizarla cualquier procesador, por lo que deben evitarse los conflictos. El planificador debe asignar los procesos preparados a los procesadores disponibles.
- **Sincronización:** puesto que hay varios procesos que pueden acceder a espacios de memoria y a recursos de E/S compartidos, debe proporcionarse una sincronización efectiva. La sincronización asegura la exclusión mutua y la ordenación de eventos.
- **Gestión de memoria:** la gestión de memoria en un multiprocesador debe comprender todos los aspectos propios de los computadores monoprocesadores, discutidos en el Capítulo 8. Además, el sistema operativo debe explotar el paralelismo que proporciona el hardware, por ejemplo las memorias multipuerto, para obtener mejores prestaciones. Los mecanismos de paginación en procesadores distintos deben coordinarse para mantener la consistencia cuando varios procesadores comparten una página o un segmento y para decidir sobre el reemplazo de páginas.
- **Fiabilidad y tolerancia a fallos:** el sistema operativo debería hacer posible una degradación gradual cuando se produce un fallo en un procesador. El planificador y otros elementos del sistema operativo deben reconocer la pérdida de un procesador y reestructurar las tablas de gestión en consecuencia.

UN SMP COMO GRAN COMPUTADOR

La mayoría de los PC y estaciones de trabajo de tipo SMP utilizan una estrategia de interconexión basada en un bus tal y como muestra la Figura 18.6. Resulta ilustrativo analizar una aproximación alternativa, que se utiliza en las implementaciones más recientes de la familia de grandes computadores (*mainframes*) IBM zSeries [SIEG04, MAK04] denominada z990. Esta familia de sistemas incluye desde computadores monoprocesador con un módulo de memoria principal hasta sistemas con 48 procesadores y ocho módulos de memoria. Los componentes clave de la configuración son los siguientes:

- **Chip con dos núcleos de procesamiento (*dual-core*):** cada microprocesador incluye dos procesadores idénticos denominados procesadores centrales (*Central Processors*, CP). Un CP es un microprocesador CISC en el que la mayoría de las instrucciones se encuentran cableadas mientras que las restante se ejecutan mediante un microcódigo vertical. Cada CP incluye cachés L1 separados para datos e instrucciones de 256 Kbytes cada una.
- **Caché L2:** cada caché L2 contiene 32 Mbytes. Las cachés L2 se organizan en grupos de cinco, de forma que cada grupo recibe accesos de ocho microprocesadores duales y proporciona acceso a todo el espacio de memoria principal.
- **Elemento de Control de Sistema (*System Control Element,* SCE):** se encarga del arbitraje de la comunicación en el sistema y tiene un papel central e el mantenimiento de la coherencia de caché.
- **Control de almacenamiento principal (*Main Store Control,* MSC):** interconecta las cachés L2 y la memoria principal.

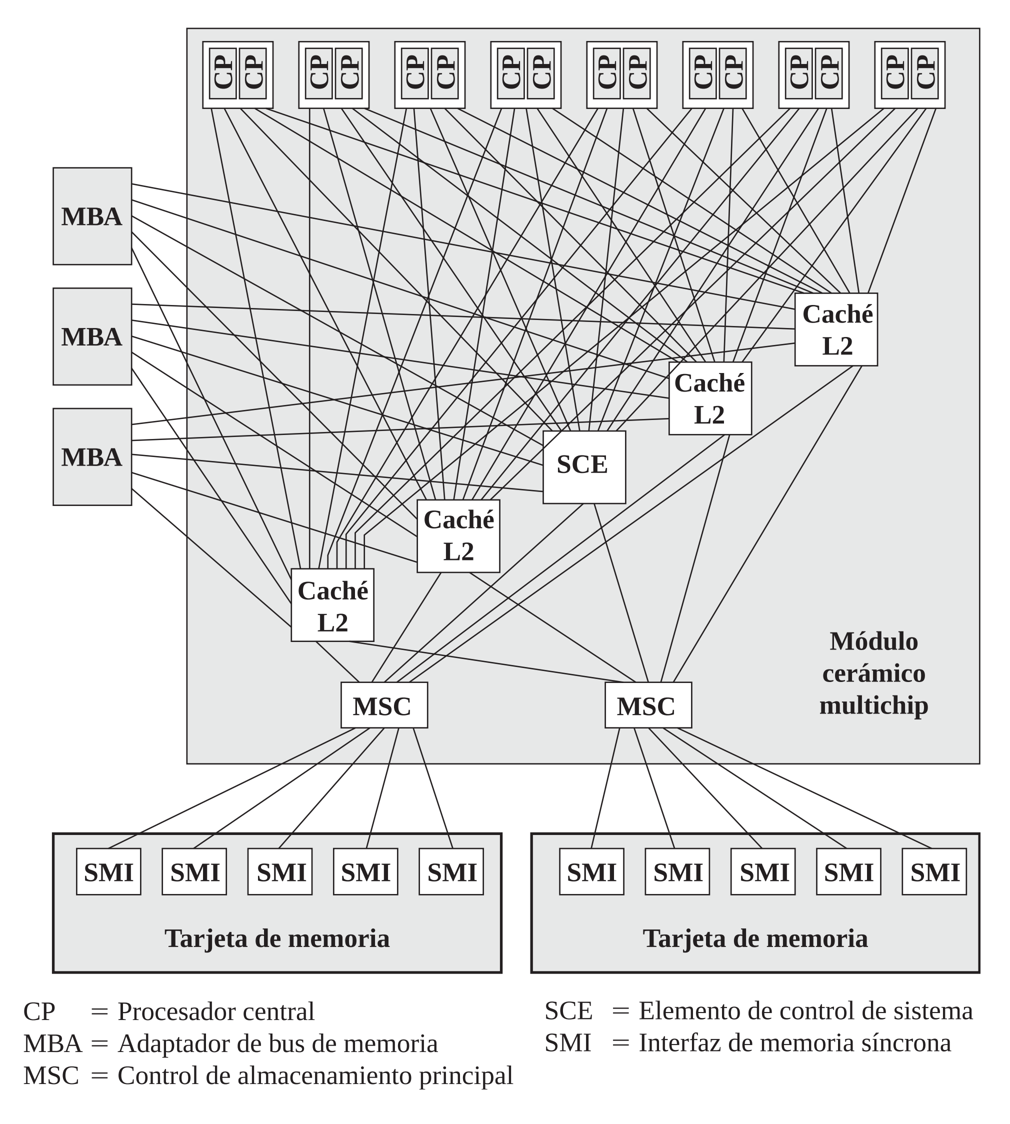


Figura 18.6. Estructura del multiprocesador IBM z990.

- **Módulos de memoria:** cada módulo dispone de 32 GB de memoria. La memoria máxima que se puede configurar consta de ocho módulos de memoria, con lo que se tiene una capacidad máxima de 256 GB. Los módulos de memoria se conectan al MSC a través de interfaces de memoria síncronos (*Synchronous Memory Interfaces*, SMI).
- **Adaptador de bus de memoria (*Memory Bus Adapter,* MBA):** el MBA proporciona una interfaz a diversos canales de E/S. El tráfico a, o desde, esos canales atraviesa directamente la caché L2.

El procesador utilizado en el z990 es relativamente peculiar si se compara con otros procesadores actuales debido a que, aunque es superescalar, ejecuta las instrucciones en el orden estricto que marca la arquitectura. Sin embargo, para compensar esto utiliza un cauce más corto y cachés y TLB mucho

mayores que otros procesadores, junto con otras características que contribuyen a mejorar las prestaciones.

El sistema z990 incluye entre uno y cuatro ***libros***. Cada libro es una unidad que consta de hasta doce procesadores y hasta 64 GB de memoria, adaptadores de E/S y un elemento de control de sistema (SCE) que los conecta. El SCE que incluye cada *libro* tiene una caché L2 de 32 MB que actúa como punto de coherencia central para ese *libro* particular. Los procesadores y los adaptadores de E/S de cada *libro* tienen acceso tanto a la caché L2 como a la memoria principal de ese *libro* o de cualquiera de los otros tres que componen el sistema. El SCE y los chips de caché L2 también se conectan con los elementos correspondientes de los otros *libros* a través de conexiones configuradas en anillo.

Hay una serie de características interesantes en el SMP z990 que pasamos a discutir:

- Interconexión conmutada.
- Cachés L2 compartidas.

Interconexión conmutada. En los PC y las estaciones de trabajo de tipo SMP es común utilizar una organización basada en un único bus compartido (Figura 18.5). Con esta organización, el bus pasa a ser un cuello de botella que afecta a la escalabilidad (escalado de las prestaciones cuando se amplía el sistema) del diseño. El z990 se enfrenta a este problema de dos formas. En primer lugar, la memoria principal se distribuye en cuatro módulos, cada una con su propio controlador de almacenamiento que puede gestionar los accesos a memoria a velocidades elevadas. El tráfico de cargas desde memoria se reduce, gracias a los caminos independientes que hay a las áreas distintas en las que se ha dividido la memoria. Cada bloque incluye dos módulos de memoria, con un total de ocho módulos en la configuración máxima. En segundo lugar, la conexión entre los procesadores (desde la caché L2) y un módulo de memoria no se lleva a cabo a través de un bus sino mediante enlaces punto a punto. Cada microprocesador tiene un enlace a cada una de las cachés L2 del *libro*, y cada caché L2 tiene un enlace, vía MSC, a cada uno de los dos módulos de memoria del mismo *libro*.

Cada caché L2 solo se conecta a los dos módulos de memoria de su *libro*. El controlador de sistema proporciona los enlaces (que no se representan en la figura) a los otros *libros* de la configuración, de forma que se pueda acceder a toda la memoria principal desde cualquier procesador.

Las conexiones a los canales de E/S también se realizan a través de enlaces punto a punto en vez de mediante buses. La caché L2 de cada *libro* se conecta al MBA de dicho libro. A su vez, el MBS se conecta a los canales de E/S.

Cachés L2 compartidas. E un esquema típico de caché de dos niveles para un SMP, cada procesador tiene una cachés L1 y L2 propias. En los últimos años, ha aumentado el interés en el concepto de utilizar una caché L2 compartida. En una de las primeras versiones de sus grandes SMP, conocida como generación 3 (G3), IBM utilizaba cachés L2 específicas para cada procesador. En las versiones más recientes (G4, G5, y serie z900) se utilizan cachés L2 compartidas. Dos consideraciones han causado este cambio:

1. En el cambio de la versión G3 a la G4, IBM pasó a utilizar microprocesadores con el doble de velocidad. Si se hubiese mantenido la organización G3, se hubiese producido un incremento significativo del tráfico a través del bus. Al mismo tiempo, se deseaba utilizar tantos

componentes de las versiones G3 como fuese posible. A no ser que se mejorase significativamente el bus, el MSC podría llegar a ser un cuello de botella.

2. El análisis de las cargas de trabajo típicas mostraba un nivel elevado de instrucciones y datos compartidos por los procesadores.

Estas consideraciones llevaron al equipo de diseño de la versión G4 a considerar el uso de una o más cachés L2 compartidas por varios procesadores (cada procesador dispone de una caché L1 interna). A primera vista, compartir la caché L2 podría parecer una mala idea. El acceso a memoria desde los procesadores podría ser más lento debido a que los procesadores deben pugnar por el acceso a la caché L2. Sin embargo, si de hecho varios procesadores comparten un elevado volumen de datos, una caché compartida puede incrementar el rendimiento en lugar de disminuirlo, ya que los datos compartidos que se encuentran en la caché compartida se obtienen más rápidamente que si se debiera acceder a ellos a través del bus.

18.3. COHERENCIA DE CACHÉ Y PROTOCOLO MESI

En los sistemas multiprocesador contemporáneos es común disponer de uno o dos niveles de caché asociados a cada procesador. Esta organización es esencial para conseguir unas prestaciones razonables. Esto, no obstante, origina un problema conocido como el problema de *coherencia de caché*. La esencia del problema es esta: pueden existir varias copias del mismo dato simultáneamente en cachés diferentes, y si los procesadores actualizan sus copias, puede producirse una visión inconsistente de la memoria. En el Capítulo 4 se definieron dos políticas de escritura usuales:

- **Postescritura (*Write back*):** las operaciones de escritura se hacen usualmente solo en la caché. La memoria principal solo se actualiza cuando la línea de caché correspondiente se reemplaza.
- **Escritura directa (*Write through*):** todas las operaciones de escritura se realizan en memoria principal a la vez que en la caché, asegurándose así que el contenido de la memoria principal siempre es válido.

Resulta evidente que una política de postescritura puede ocasionar inconsistencia. Si dos cachés contienen la misma línea, y la línea se actualiza en una caché, la otra caché tendrá un valor no válido. Las lecturas siguientes a dicha línea producirán resultados no válidos. Incluso con la política de escritura directa puede existir inconsistencia, a no ser que las otras cachés comprueben los accesos a la memoria principal o reciban algún tipo de notificación directa de la escritura realizada.

En esta sección, revisaremos brevemente distintas aproximaciones al problema de la coherencia de caché, y después nos centraremos en la aproximación más ampliamente utilizada: el protocolo MESI. Una versión de este protocolo se utiliza tanto en implementaciones del Pentium 4 como del PowerPC.

El objetivo de un protocolo de coherencia de caché es situar las variables locales utilizadas recientemente en la caché apropiada y mantenerlas allí para las distintas escrituras y lecturas, al mismo tiempo que se mantiene la consistencia de las variables compartidas que pudieran encontrarse en varias cachés al mismo tiempo. Las aproximaciones de coherencia de caché generalmente se han dividido en aproximaciones software y hardware. Algunas implementaciones adoptan una estrategia que implica tanto elementos software como hardware. No obstante, la distinción entre aproximaciones

software y hardware es todavía instructiva y es comúnmente utilizada en la presentación de las estrategias de coherencia de caché.

SOLUCIONES SOFTWARE

Los esquemas software de coherencia de caché intentan evitar la necesidad de circuitería y lógica hardware adicional dejando que el compilador y el sistema operativo se encarguen del problema. Las propuestas software son atractivas porque transfieren el costo de la detección de posibles problemas desde el hardware al software. Por otra parte, en el momento de la compilación, el software generalmente debe tomar ciertas decisiones conservadoras que pueden ocasionar una utilización ineficiente de la caché.

Los mecanismos de coherencia basados en el compilador realizan un análisis del código para determinar qué datos pueden dar problemas al pasar a caché, y los marcan en consecuenca. Después, el sistema operativo o el hardware impiden que se pasen a caché los datos marcados como no almacenables en caché (*non-cachéable*).

El enfoque más sencillo consiste en impedir que cualquier dato compartido pase a caché. Esto es demasiado conservador puesto que una estructura de datos compartida puede utilizarse de manera exclusiva durante ciertos periodos de tiempo y solo para lectura en otros periodos. Es solo durante aquellos periodos en los que al menos un procesador pueda actualizar una variable y otro procesador pueda acceder a la variable cuando hay que considerar la coherencia de caché.

Hay aproximaciones más eficientes que analizan el código y determinan periodos seguros para las variables compartidas. El compilador inserta entonces instrucciones en el código generado para reforzar la coherencia de caché en los periodos críticos. Se han desarrollado cierto número de técnicas para realizar el análisis y para mejorar los resultados; *véanse* las revisiones de [LIL93] y [STEN90].

SOLUCIONES HARDWARE

Las soluciones basadas en el hardware generalmente se denominan protocolos de coherencia de caché. Estas soluciones permiten reconocer dinámicamente en el momento de la ejecución las situaciones de inconsistencias potenciales. Puesto que el problema se considera solo en el momento en que aparece, existe un uso más efectivo de las cachés, mejorándose las prestaciones en relación a las aproximaciones software. Además, estas aproximaciones son transparentes para el programador y el compilador, reduciendo la complejidad en el desarrollo del software.

Los esquemas hardware difieren en una serie de aspectos, que incluyen el lugar donde se encuentra la información de estado de las líneas de datos, cómo se organiza esa información, dónde se impone la coherencia y los mecanismos para imponerla. En general, los esquemas hardware se pueden dividir en dos categorías: protocolos de directorio y protocolos de sondeo (*snoopy protocols*).

Protocolos de directorio. Los protocolos de directorio recogen y mantienen la información acerca de dónde residen las copias de las líneas. Usualmente, hay un controlador centralizado que es parte del controlador de memoria principal, y un directorio que se almacena en la memoria principal. El directorio contiene información de estado global en relación con los contenidos de las diferentes cachés locales. Cuando el controlador individual de una caché hace una petición, el controlador centralizado comprueba y emite las órdenes precisas para la transferencia entre memoria y caché o entre distintas cachés. Además, es responsable de mantener actualizada la información de estado; de esta

forma, cualquier acción local que pueda afectar al estado global de una línea debe comunicarse al controlador central.

Normalmente, el controlador posee información acerca de los procesadores que tienen una copia de cada línea. Antes de que un procesador pueda escribir en una copia local de una línea, debe solicitar al controlador el acceso exclusivo a dicha línea. Antes de ceder este acceso exclusivo, el controlador envía un mensaje a todos los procesadores con una copia de la línea en su caché, forzando a que cada procesador invalide su copia. Después de recibir el reconocimiento de cada uno de esos procesadores, el controlador cede el acceso exclusivo al procesador que lo solicitó. Cuando otro procesador intenta leer una línea cedida para acceso exclusivo de otro procesador, enviará una notificación de fallo de caché al controlador. Entonces el controlador manda una orden al procesador que posee la línea requerida para que lo vuelva a escribir en memoria principal. Ahora, la línea puede compartirse para lectura por el procesador original y el que solicitaba el acceso.

Los esquemas de directorio presentan las desventajas propias de tener un cuello de botella central y del coste de comunicación entre los controladores de las distintas cachés y el controlador central. No obstante, son efectivos en sistemas de gran escala que poseen múltiples buses o algún esquema complejo de interconexión.

PROTOCOLOS DE SONDEO (*SNOOPY PROTOCOLS*)

Los protocolos de sondeo distribuyen la responsabilidad de mantener la coherencia de caché entre todos los controladores de caché del multiprocesador. Una caché debe reconocer cuando una línea de las que contiene está compartida con otras cachés. Cuando se realiza una actualización en una línea de caché compartida, debe anunciarse a todas las otras cachés mediante un mecanismo de difusión (*broadcast*). Cada controlador de caché es capaz de sondear o «espiar» (*«snoop»*) la red para observar las notificaciones que se difunden, y reaccionar adecuadamente.

Los protocolos de sondeo se adaptan bien a los multiprocesadores basados en un bus, puesto que el bus compartido proporciona una forma sencilla para la difusión y el sondeo. No obstante, puesto que uno de los objetivos de utilizar cachés locales es evitar los accesos al bus, hay que cuidar que el incremento en el tráfico del bus que requiere la difusión y el sondeo no anule los beneficios de las cachés locales.

Se han explorado dos enfoques básicos del protocolo de sondeo: invalidar-si-escritura (*write-invalidate*) y actualizar-si-escritura, o difundir-escritura (*write-update* o *write-broadcast*). Con un protocolo de invalidar-si-escritura, puede haber múltiples procesadores que leen pero un solo procesador que escribe en un momento dado. Inicialmente, una línea puede compartirse por varias cachés con el propósito de lectura. Cuando se quiere hacer una escritura en la línea de una caché, primero envía una notificación que invalida la línea en las otras cachés, haciendo que dicha línea sea exclusiva para la caché donde se va a escribir. Una vez que la línea es exclusiva, el procesador puede realizar escrituras locales en la misma hasta que otro procesador solicita la misma línea.

Con un protocolo de actualizar-si-escritura, puede haber varios procesadores que escriben igual que varios procesadores que leen. Cuando un procesador desea escribir en una línea compartida, la palabra a actualizar se distribuye a los demás, y las cachés que contienen esa línea lo pueden actualizar.

Ninguna de estas dos aproximaciones es mejor que la otra en todas las situaciones. Las prestaciones dependen del número de cachés locales y del patrón de escrituras y lecturas de memoria.

Algunos sistemas implementan protocolos adaptativos que utilizan tanto el mecanismo de invalidar-si-escritura como el de actualizar-si-escritura.

La aproximación de invalidar-si-escritura es la que más se utiliza en los multiprocesadores comerciales, tales como los basados en el Pentium II y en el PowerPC. Se marca el estado de cada línea de caché (usando dos bits adicionales en el campo de marca de la caché) como modificado (*modified*), exclusivo (*exclusive*), compartido (*shared*) o no-válido (*invalid*). Por esta razón, el protocolo de invalidar-si-escritura se llama MESI, de las iniciales de los estados en inglés. En el Capítulo 4 ya consideramos el protocolo MESI en relación con el protocolo de coordinación entre los niveles 1 y 2 de cachés locales. En el resto de esta sección, consideraremos su uso para las cachés locales en un multiprocesador. Por simplicidad en la presentación no se examinan los mecanismos necesarios para la coordinación local del nivel 1 y el 2 al mismo tiempo que los de coordinación en el multiprocesador distribuido. Esto no añade ningún concepto nuevo y en cambio complicaría considerablemente la discusión.

EL PROTOCOLO MESI

Para proporcionar coherencia de caché en un SMP, la caché de datos usualmente implementa un protocolo conocido como MESI. Con el protocolo MESI, la caché de datos incluye dos bits de estado en la marca, puesto que cada línea puede estar en uno de estos cuatro estados:

- **Modificado (*Modified*):** la línea de caché ha sido modificada (es distinta a su valor en memoria principal) y está disponible solo en esta caché.
- **Exclusivo (*Exclusive*):** la línea de caché tiene el mismo contenido que en memoria principal y no está presente en ninguna otra caché.
- **Compartido (*Shared*):** la línea de caché tiene el mismo contenido que en memoria principal y puede estar presente en otra caché.
- **No-Válido (*Invalid*):** la línea de caché no contiene datos válidos.

La Tabla 18.1 resume el significado de los cuatro estados. La Figura 18.7 muestra el diagrama de estados del protocolo MESI. Recuérdese que cada línea de caché tiene sus propios bits de estado y su

Tabla 18.1. Estados MESI para las líneas de caché.

	M Modificado	E Exclusivo	S Compartido	I No-válido
¿Línea de caché válida?	Sí	Sí	Sí	No
La copia de memoria es...	atrasada	válida	válida	—
¿Existen copias en otras cachés?	No	No	Quizá	Quizá
Una escritura en esta línea...	no va al bus	no va al bus	va al bus y actualiza la caché	va directamente al bus

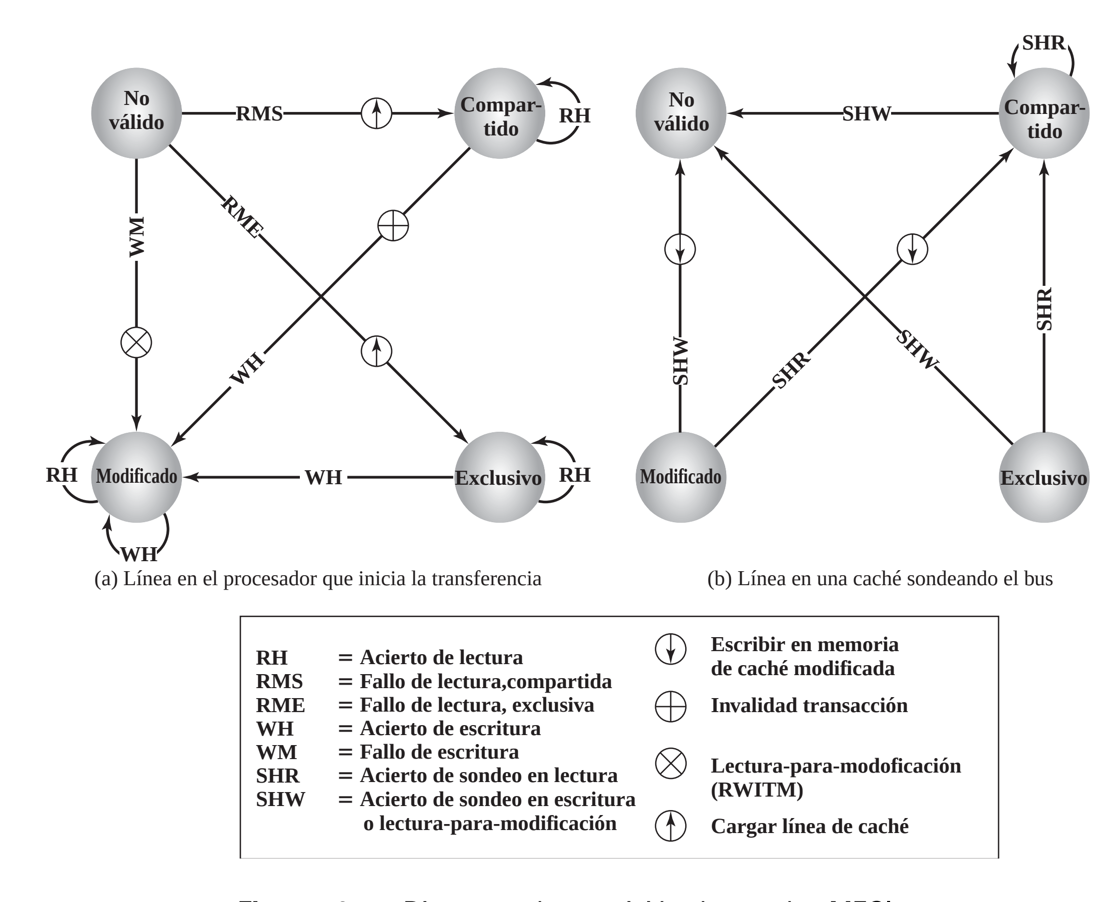


Figura 18.7. Diagrama de transición de estados MESI.

propia implementación del diagrama de estados. La Figura 18.7a muestra las transiciones que se producen debido a las acciones iniciadas por el procesador al que pertenece la caché. La Figura 18.7b describe las transiciones ocasionadas por eventos que se producen en el bus común. La presentación de las acciones ocasionadas por el procesador en un diagrama de estado distinto al utilizado para las acciones ocasionadas por eventos del bus, ayuda a clarificar la lógica del protocolo MESI. Sin embargo, en cada instante una línea de caché está en un único estado. Si el siguiente evento proviene del procesador al que pertenece la caché, entonces la transición se describe en la Figura 18.7a, y si el siguiente evento proviene del bus, la transición se describe en la Figura 18.7b. A continuación se describen cada una de las transiciones con más detalle.

Fallo de lectura. Cuando se produce un fallo de lectura en la caché local, el procesador inicial una lectura en memoria para acceder a la línea de memoria principal que contiene la dirección que no está en caché. El procesador inserta una señal en el bus que avisa a todos los otros procesadores/cachés para que sondeen la transacción. Hay una serie de posibilidades:

- Si una de las cachés tiene una copia limpia (*clean*) de la línea (es decir, una copia no modificada desde que se leyó de memoria) en el estado exclusivo, devuelve una señal indicando que comparte la línea. El procesador que envía esta señal pasa su copia al estado compartido y el procesador inicial lee la línea y pasa el estado de esta en su caché de no-válido a compartido.

- Si una o más cachés tienen una copia limpia de la línea en estado compartido, cada una de ellas indica que comparte la línea. El procesador inicial lee la línea y pasa su estado en caché de no válido a compartido.
- Si una de las cachés tiene una copia modificada de la línea, entonces esa caché bloquea la lectura de memoria y proporciona la línea a la caché que la solicita a través del bus compartido. La caché que proporciona la línea pasa esta del estado modificado al estado compartido[1].
- Si ninguna otra caché tiene una copia de la línea (limpia o en estado modificado), no se envía ninguna señal. El procesador lee la línea y cambia el estado de la línea en su caché de no válido a exclusivo.

Acierto de lectura. Cuando se produce un acierto en una lectura dentro de una línea presente en la caché local, el procesador simplemente lee el dato solicitado. No hay ningún cambio de estado: se mantiene como modificado, compartido o exclusivo.

Fallo de escritura. Cuando se produce un fallo en una escritura en la caché local, el procesador comienza una lectura de memoria para acceder a la línea de memoria principal que contiene la dirección que no está en caché. Para ello, el procesador envía una señal a través del bus que indica lectura-para-modificación (RWITM, *Read-With-Intent-To-Modify*). Cuando se carga la línea, se marca inmediatamente como modificada. Con respecto a las otras cachés, hay dos escenarios posibles previos a la carga del bloque de datos.

Primero, otro procesador puede haber modificado una copia de esta línea (estado modificado) en su caché. En este caso, el procesador en cuestión indica al procesador inicial que hay una copia modificada de la línea. El procesador inicial deja libre el bus y espera. El procesador con la copia modificada accede al bus, escribe la línea modificada en la memoria principal y cambia el estado de la línea en caché a no válido (puesto que el procesador inicial va a modificar esta línea). A continuación, el procesador inicial activará una señal RWITM en el bus y luego lee la línea de la memoria principal, modifica la línea en la caché y la marca con el estado modificado.

El segundo escenario corresponde a que ninguna caché tiene una copia modificada de la línea en su caché. En este caso no se responde con ninguna señal, y el procesador inicial prosigue leyendo la línea y modificándola. Mientras tanto, si una o más cachés tienen copias limpias de la línea en estado compartido, cada caché pasa su copia a estado no válido. Si una caché tiene una copia de la línea no modificada en estado exclusivo, la invalida.

Acierto de escritura. Cuando se produce un acierto de escritura en una línea de caché local, el efecto depende del estado de la línea:

- **Compartido:** antes de realizar la actualización, el procesador debe conseguir el acceso exclusivo a la línea. El procesador señala esta intención a través del bus. Todo procesador que tenga una copia de la línea en su caché la cambia del estado compartido al no válido. Después el

[1] En algunas implementaciones, la caché con la línea modificada indica al procesador que inició el acceso que vuelva a intentarlo. Mientras tanto, el procesador con la copia modificada accede al bus, actualiza la línea modificada en memoria principal y pasa la línea desde el estado modificado al compartido. Después, el procesador que solicitó el acceso intenta de nuevo y encuentra que uno o varios procesadores tienen una copia actualizada de la línea en estado compartido, como se describe en el punto precedente.

procesador inicial actualiza la línea y cambia su copia de la línea del estado compartido al estado modificado.

- **Exclusivo:** puesto que el procesador tiene el control exclusivo de esta línea, simplemente la actualiza y cambia el estado de la línea de exclusivo a modificado.
- **Modificado:** puesto que el procesador tiene el control exclusivo de la línea y la tiene marcada en el estado modificado, solo tiene que actualizarla.

Consistencia de cachés L1-L2. Hasta ahora se han descrito los protocolos de coherencia de caché en términos de la cooperación entre las cachés conectadas al mismo bus o a otro sistema de interconexión utilizado por un SMP. Normalmente, estas cachés son cachés L2, y cada procesador además tiene una caché L1 que no está conectada directamente al bus y que por lo tanto no puede participar en un protocolo de sondeo. De esta forma, se necesita un esquema para mantener la integridad de los datos en los dos niveles de caché y en todas las cachés del SMP.

La estrategia utilizada consiste en extender el protocolo MESI (o cualquier otro protocolo de coherencia de caché) a las cachés L1. Así, cada línea de la caché L1 incluye bits para indicar el estado. Básicamente, el objetivo es el siguiente: para cualquier línea presente tanto en una caché L2 como en su correspondiente caché L1, el estado de la línea en L1 debe seguir el trayecto del estado de la línea en L2. Una forma sencilla de hacer esto es adoptar una política de escritura directa en la caché L1, pero escribiendo sobre la caché L2 y no sobre la memoria principal. La política de escritura directa en L1 hace que cualquier modificación en una línea de L1 se propague a la caché L2, haciéndose visible a las otras cachés L2. El uso de la política de escritura directa en L1 requiere que el contenido de L1 sea un subconjunto del contenido de L2. Esto sugiere además que la asociatividad de L2 debería ser igual o mayor que la asociatividad de L1. La política de escritura directa para L1 se utiliza en el SMP IBM S/390.

Si la caché L1 utiliza una política de post escritura, la relación entre las dos cachés es más compleja. Hay varias aproximaciones para mantener la coherencia en este caso. El procedimiento utilizado en el Pentium II se describe con detalle en [SHAN05].

18.4. PROCESAMIENTO MULTIHEBRA Y MULTIPROCESADORES MONOCHIP

La medida de prestaciones más importante para un procesador es la velocidad a la que ejecuta instrucciones. Esta se puede expresar como

$$\text{velocidad } (MIPS) = f \times IPC$$

donde f es la frecuencia de reloj del procesador, en MHz e IPC (instrucciones por ciclo) es el promedio de instrucciones ejecutadas por ciclo. En consecuencia, los diseñadores han buscado la mejora de prestaciones en dos frentes: incrementando la frecuencia de reloj e incrementando el número de instrucciones ejecutadas o, más propiamente, el número de instrucciones que se completan en un ciclo de reloj. Como se ha visto en capítulos anteriores, los diseñadores han aumentado el valor de IPC utilizando un cauce segmentado y después utilizando varios cauces paralelos en las arquitecturas superescalares. Con los diseños segmentados, el principal problema es maximizar la utilización de cada etapa del cauce. Para aumentar el rendimiento, los diseñadores han ideado mecanismos bastante

complejos, tales como la ejecución de instrucciones en un orden diferente al que aparecen en el código y el inicio de la ejecución de instrucciones que puede que no se tengan que ejecutar. Sin embargo, como se indicó en la Sección 2.2, esta aproximación puede estar llegando a su límite debido a los problemas relacionados con el aumento de la complejidad y el consumo de potencia.

Una alternativa, que permite una paralelismo entre instrucciones elevado sin incrementar ni la complejidad de los circuitos ni el consumo de potencia, es el procesamiento multihebra (*multithreading*). Básicamente, la secuencia de instrucciones se divide en secuencias más pequeñas, denominadas hebras (*threads*), que pueden ejecutarse en paralelo.

La diversidad de diseños multihebra que se han realizado, tanto en procesadores comerciales como experimentales, es considerable. En esta sección, proporcionamos una breve revisión de los conceptos principales.

PROCESAMIENTO MULTIHEBRA IMPLÍCITO Y EXPLÍCITO

El concepto de hebra utilizado para estudiar los procesadores multihebra puede ser o puede no ser el mismo que el concepto de hebra en un sistema operativo multiprogramado. Por lo tanto, es útil definir brevemente una serie de términos:

- **Proceso:** un programa en ejecución en un computador. Un proceso reúne dos características clave:
  - **Propiedad de recursos:** un proceso dispone de un espacio de direcciones virtuales para almacenar la imagen del proceso, que consta del programa, los datos, la pila y demás atributos que definen el proceso. En determinadas ocasiones, un proceso puede tener el control o poseer recursos tales como memoria principal, canales de E/S, dispositivos de E/S y ficheros.
  - **Planificación/ejecución:** la ejecución de un proceso sigue un camino de ejecución (traza) a través de uno o más programas. Esta ejecución puede entremezclarse con la de otros procesos. Así, un proceso tiene un estado de ejecución (preparado, en ejecución, etc.) y una prioridad de asignación, y es la entidad que el sistema operativo se encarga de planificar y asignar.
- **Conmutación de proceso:** operación que cambia el proceso que se está ejecutando en el procesador por otro proceso. Para ello, almacena todos los datos de control, registros y demás información del primer proceso y los reemplaza con la información correspondiente al segundo[2].
- **Hebra:** una unidad de trabajo dentro de un proceso que se puede asignar al procesador. Incluye un contexto de procesador (con el contador de programa y el puntero de pila) y su propia área de datos para la pila (para permitir las llamadas a subrutinas). Una hebra se ejecuta secuencialmente y puede interrumpirse para que el procesador pase a ejecutar otra hebra.

---

[2] El término cambio de contexto (*context switching*) aparece frecuentemente en la literatura y en los libros de texto dedicados a sistemas operativos. Desafortunadamente, aunque en la mayoría de la literatura se utiliza el término para indicar lo que aquí hemos denominado cambio o conmutación de proceso, otras fuentes lo utilizan para referirse a una conmutación de hebras. Para evitar ambigüedades, en este libro no utilizamos este término.

- **Conmutación de hebra:** el control del procesador pasa de una hebra a otra dentro de un mismo proceso. Usualmente, este tipo de conmutación es mucho menos costosa que la de procesos.

Así, mientras que las hebras aparecen involucradas en la planificación y ejecución, los procesos tienen que ver tanto con la planificación/ejecución como con la asignación de recursos. Las hebras de un mismo proceso comparten los mismos recursos. Esta es la razón por la que un cambio de hebra consume mucho menos tiempo que la conmutación de procesos. Los sistemas operativos tradicionales, tales como las primeras versiones de Unix, no utilizaban hebras. La mayoría de los sistemas operativos actuales, tales como Linux, otras versiones de Unix y Windows, sí permiten el uso de hebras. Se puede distinguir entre hebras de nivel de usuario (*user-level threads*), visibles para los programas de aplicación, y hebras de nivel de núcleo (*kernel-level threads*), visibles solo para el sistema operativo. Ambos tipos de hebras se pueden denominar hebras explícitas, definidas en el software.

Hasta ahora, todos los procesadores comerciales y la mayoría de los experimentales han utilizado procesamiento multihebra explícito. Estos sistemas ejecutan concurrentemente instrucciones de hebras explícitas diferentes, bien entremezclando instrucciones de hebras diferentes en cauces compartidos o mediante ejecución paralela y cauces paralelos. El procesamiento multihebra implícito hace referencia a la ejecución concurrente de varias hebras extraídas de un único programa secuencial. Estas hebras implícitas pueden ser definidas estáticamente por el compilador o dinámicamente por el hardware. En el resto de esta sección consideraremos el procesamiento multihebra explícito.

APROXIMACIONES AL PROCESAMIENTO MULTIHEBRA EXPLÍCITO

Como mínimo, un procesador multihebra debe proporcionar un contador de programa distinto para cada una de las hebras que puedan ejecutarse concurrentemente. Los diseños cambian según la cantidad y el tipo de hardware que se añade para permitir la ejecución concurrente de las hebras. En general, se captan instrucciones para cada hebra. El procesador trata cada hebra separadamente y puede utilizar técnicas como la predicción de saltos, el renombramiento de registros u otras técnicas superescalares para optimizar la ejecución de una hebra. Lo que se aprovecha es un paralelismo entre hebras (*thread-level parallelism*) que, sumado al paralelismo entre instrucciones, puede proporcionar una mejora de prestaciones considerable.

En un sentido amplio, se pueden considerar cuatros aproximaciones prinipales al procesamiento multihebra:

- **Multihebra entrelazada:** también se conoce con el nombre de **procesamiento multihebra de grano fino**. El procesador trabaja con dos o más contextos al mismo tiempo, conmutando entre uno y otro en cada ciclo de reloj. Si una hebra se bloquea debido a dependencias de datos o retardos de memoria, esa hebra se salta y se pasa a ejecutar una hebra que esté preparada.
- **Multihebra con bloqueo:** también se conoce como procesamiento multihebra de grano grueso. Las instrucciones de una hebra se ejecutan sucesivamente hasta que se produce un evento que puede ocasionar un retardo, tal como un fallo de caché. Este evento induce una conmutación a otra hebra. Esta aproximación es efectiva en un procesador con ejecución ordenada en cuyo cauce se produciría un atasco debido al retardo asociado a un evento como un fallo de caché.
- **Multihebra simultánea (SMT):** instrucciones correspondientes a hebras diferentes se emiten simultáneamente a las unidades funcionales de un procesador superescalar. Se combina así la

capacidad de emisión de varias instrucciones del procesador superescalar y el uso de múltiples contextos correspondientes a las diferentes hebras.

- **Multiprocesador monochip**: en este caso, existen varias copias del procesador en un solo circuito integrado y cada procesador actúa sobre hebras diferentes. La ventaja de esta aproximación es que el área del circuito integrado se utiliza eficientemente evitando la cada vez mayor complejidad de los nuevos diseños del cauce segmentado.

En las dos primeras aproximaciones, las instrucciones que pertenecen a hebras distintas no se ejecutan simultáneamente. En su lugar, el procesador es capaz de conmutar rápidamente de una hebra a otra, utilizando un conjunto de registros diferentes y otra información de contexto. Esto permite un mejor aprovechamiento de los recursos de ejecución del procesador y evita las elevadas penalizaciones asociadas a los fallos de caché u otros eventos con retardos asociados. La alternativa SMT supone una ejecución simultánea real de instrucciones de hebras diferentes, utilizando los recursos de ejecución repetidos. Los multiprocesadores monochip también permiten la ejecución simultánea de instrucciones de hebras diferentes.

La Figura 18.8, basada en una de [UNGE02], muestra algunas de las posibles arquitecturas de cauce que implican procesamiento multihebra, y las compara con alternativas que no utilizan procesamiento multihebra. Cada fila horizontal representa el hueco o huecos de emisión posibles por ciclo de ejecución; es decir, la anchura de cada fila corresponde al máximo número de instrucciones que pueden emitirse en un ciclo de reloj[3]. La dimensión vertical representa la secuencia temporal de ciclos de reloj. Un hueco vacío (sombreado) significa un hueco de ejecución no utilizado en el cauce. Una no operación se indica con N.

Las primeras tres ilustraciones de la Figura 18.8 muestran distintas aproximaciones con un procesador escalar (es decir, con emisión de una única instrucción):

- **Escalar de una hebra**: este el tipo de cauce simple que se encuentra en las máquinas RISC y CISC tradicionales, sin procesamiento multihebra.
- **Escalar multihebra con entrelazado:** esta es la aproximación al procesamiento multihebra más fácil de implementar. Al conmutar de una hebra a otra en cada ciclo, las etapas del cauce pueden mantenerse totalmente ocupadas, o casi totalmente ocupadas. El hardware debe ser capaz de conmutar entre contextos de hebras entre ciclos.
- **Escalar multihebra con bloqueo:** en este caso, cada hebra se ejecuta hasta que se produce un evento que ocasiona un retardo que podría detener el cauce, y que hace que el procesador conmute a otra hebra.

La Figura 18.8c muestra una situación en la que el tiempo necesario para realizar el cambio de hebra es igual a un ciclo, mientras que en la Figura 18.8b la conmutación entre hebras no consume ningún ciclo de reloj. En este caso, se asume que no hay control de dependencias de datos en el procesamiento multihebra entrelazado, con lo cual el diseño del cauce se simplifica y se podría conseguir

[3] Los huecos de emisión son las posiciones desde las que las instrucciones pueden emitirse en un ciclo de reloj. Recuérdese del Capítulo 14 que la emisión de una instrucción es el proceso de iniciar la ejecución de una instrucción en una unidad funcional del procesador. Esto se produce cuando la instrucción se transfiere desde la etapa de decodificación del cauce a la primera etapa de ejecución del cauce.

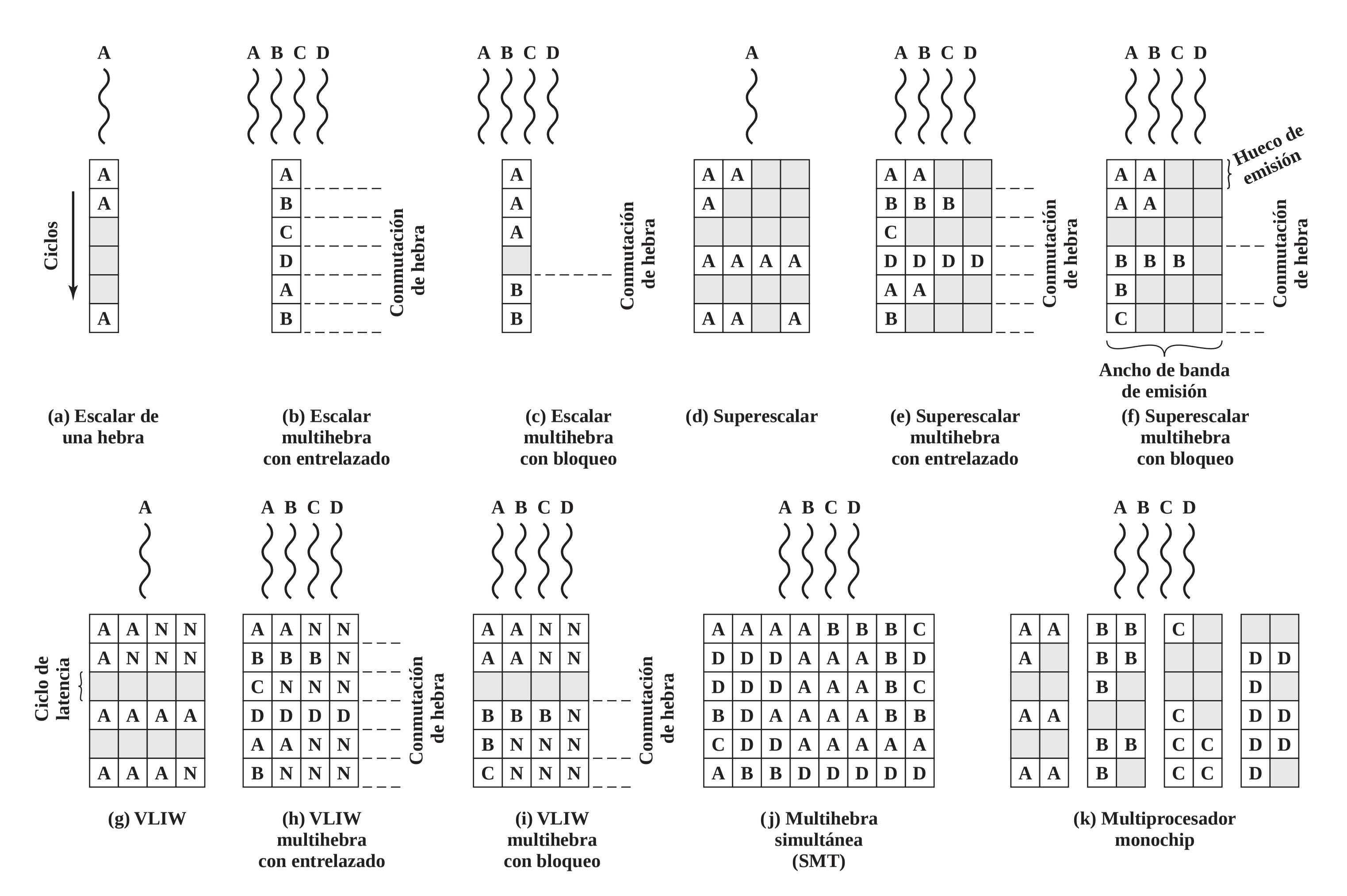


Figura 18.8. Alternativas para la ejecución de varias hebras.

la conmutación de hebras sin retardos. Sin embargo, según el diseño y la implementación concreta que se tenga, el procesamiento multihebra con bloqueo puede necesitar un ciclo de reloj para conmutar entre hebras, como se muestra en la Figura 18.8. Esto es así si una instrucción captada activa la conmutación de hebra y la hebra a la que pertenece dicha instrucción debe sacarse del cauce [UNGE03].

Aunque el procesamiento multihebra entrelazado parece proporcionar una mejor utilización del procesador que el procesamiento multihebra con bloqueo, esto se consigue sacrificando las prestaciones del procesamiento de cada hebra. Las distintas hebras compiten por la caché, dando lugar a un aumento de la probabilidad de fallo de caché para cada una de las hebras.

Si el procesador puede emitir varias instrucciones por ciclo, aumentan las oportunidades de ejecución paralela. Las Figuras 18.8d y 18.8i muestran algunas diferencias entre procesadores que disponen de hardware para emitir cuatro instrucciones por ciclo. En todos estos casos, en cada ciclo únicamente se emiten instrucciones de una misma hebra. Las alternativas que se muestran son las siguientes:

- **Superescalar:** es la aproximación superescalar básica sin procesamiento multihebra. Hasta hace relativamente poco, esta era la aproximación más potente para aprovechar el paralelismo dentro del procesador. Hay que tener en cuenta que, en algunos ciclos, no todos huecos de emisión se utilizan. En esos ciclos, se emiten menos instrucciones que el número máximo permitido, lo que se denomina *pérdida horizontal*. Durante otros ciclos de instrucción no se utiliza ningún hueco de emisión. Esto significa que no se emite ninguna instrucción y se denomina *pérdida vertical*.
- **Superescalar multihebra con entrelazado:** en cada ciclo, se emiten tantas instrucciones como se pueda de una única hebra. Con esta técnica se eliminan los posibles retardos asociados las conmutaciones entre hebras, como se han descrito anteriormente. No obstante, el número de instrucciones que se emite en cada ciclo sigue estando limitado por las dependencias que existen dentro de una hebra.
- **Superescalar multihebra con bloqueo:** de nuevo, solo instrucciones de una hebra pueden emitirse en un ciclo, y se utiliza procesamiento multihebra con bloqueo.
- **Procesadores de palabra de instrucciones muy larga (VLIW, *Very Long Instruction Word*):** una arquitectura VLIW, tal como el la IA-64, ubica varias instrucciones en una única palabra. Usualmente, el compilador construye los códigos VLIW ubicando operaciones que pueden ejecutarse en paralelo dentro de la misma palabra. En una máquina VLIW sencilla (Figura 18.8g) si no es posible rellenar completamente la palabra con instrucciones que puedan emitirse en paralelo, se utilizan códigos de no operar (*no-ops*).
- **VLIW multihebra con entrelazado:** esta aproximación proporcionaría eficiencias similares a las del procesamiento multihebra con entrelazado en una arquitectura superescalar.
- **VLIW multihebra con bloqueo:** esta aproximación proporcionaría eficiencias similares a las del procesamiento multihebra con bloqueo en una arquitectura superescalar.

Las dos últimas aproximaciones que se muestran en la Figura 18.8 permiten la ejecución simultánea y paralela de varias hebras:

- **Procesadores multihebra simultánea (SMT):** la Figura 18.8i muestra un sistema capaz de emitir ocho instrucciones a la vez. Si una hebra tiene un mayor grado de paralelismo entre

instrucciones podría, en algunos ciclos, llenar todos los huecos de emisión horizontales. En otros ciclos, se podrían emitir instrucciones de dos o más hebras. Si hay suficientes hebras activas, usualmente sería posible emitir el máximo número de instrucciones en cada ciclo, proporcionando un alto nivel de eficiencia.

- **Multiprocesadores monochip:** la Figura 18.8k correspondería a un chip con cuatro procesadores, cada uno de los cuales es un procesador superescalar capaz de emitir dos instrucciones. A cada uno de los procesadores se le asigna una hebra, de la que puede emitir hasta dos instrucciones por ciclo.

Comparando las Figuras 18.8j y 18.8k, vemos que un multiprocesador monochip con la misma capacidad de emisión que un SMT no es capaz de conseguir el mismo grado de paralelismo entre instrucciones. Esto se debe a que el multiprocesador monochip no puede ocultar las latencias emitiendo instrucciones de otras hebras. No obstante, el multiprocesador monochip mejoraría las prestaciones de un superescalar con la misma capacidad de emisión de instrucciones, debido a que las pérdidas horizontales serían mayores en el procesador superescalar. Además, es posible utilizar procesamiento multihebra dentro de cada uno de los procesadores del multiprocesador monochip, y esto es lo que se hace en algunas de las máquinas actuales.

EJEMPLOS DE SISTEMAS

Pentium 4. Los modelos más recientes del Pentium 4 utilizan una técnica de procesamiento multihebra que, en la literatura de Intel, se denomina *hyperthreading* [MARR02]. Básicamente, la aproximación del Pentium 4 corresponde a un SMT que permite procesar dos hebras. De esta forma, un único procesador multihebra equivaldría a dos procesadores lógicos.

IBM Power5. El chip Power5 de IBM, que se utiliza en los productos PowerPC de la gama alta, combina multiprocesamiento monochip con SMT [KALL04]. El chip tiene dos procesadores, cada uno de los cuales es un procesador multihebra capaz de procesar dos hebras concurrentemente utilizando SMT. Es interesante resaltar que los diseñadores simularon varias alternativas y encontraron que dos procesadores SMT de dos vías en un chip proporcionaban mejores prestaciones que un solo procesador SMT de cuatro vías. Las simulaciones pusieron de manifiesto que el procesamiento multihebra de más de dos hebras podría ocasionar una reducción de prestaciones debido a la interacción por los accesos a caché, dado que los datos que necesita una hebra podrían desplazar de caché a los datos que necesita otra hebra.

La Figura 18.9 muestra el diagrama de flujo de instrucciones del Power5. Solo deben repetirse algunos de los elementos del procesador para disponer de elementos dedicados a cada una de las hebras. Se utilizan dos contadores de programa. El procesador alterna la captación de instrucciones, hasta ocho cada vez, entre las dos hebras. Todas las instrucciones se almacenan en una caché de instrucciones común y comparten un recurso de traducción de instrucciones que lleva a cabo una decodificación de instrucciones parcial. Cuando aparece un salto condicional, el recurso de predicción de salto predice la dirección del mismo y, si es posible, calcula la dirección de destino del salto. Para predecir el destino de un retorno de subrutina, el procesador utiliza una pila (pila de retorno) para cada hebra.

Después, las instrucciones pasan a dos buffers de instrucciones separados. Entonces, según la prioridad de la hebra, se selecciona un grupo de instrucciones y se decodifica en paralelo. A

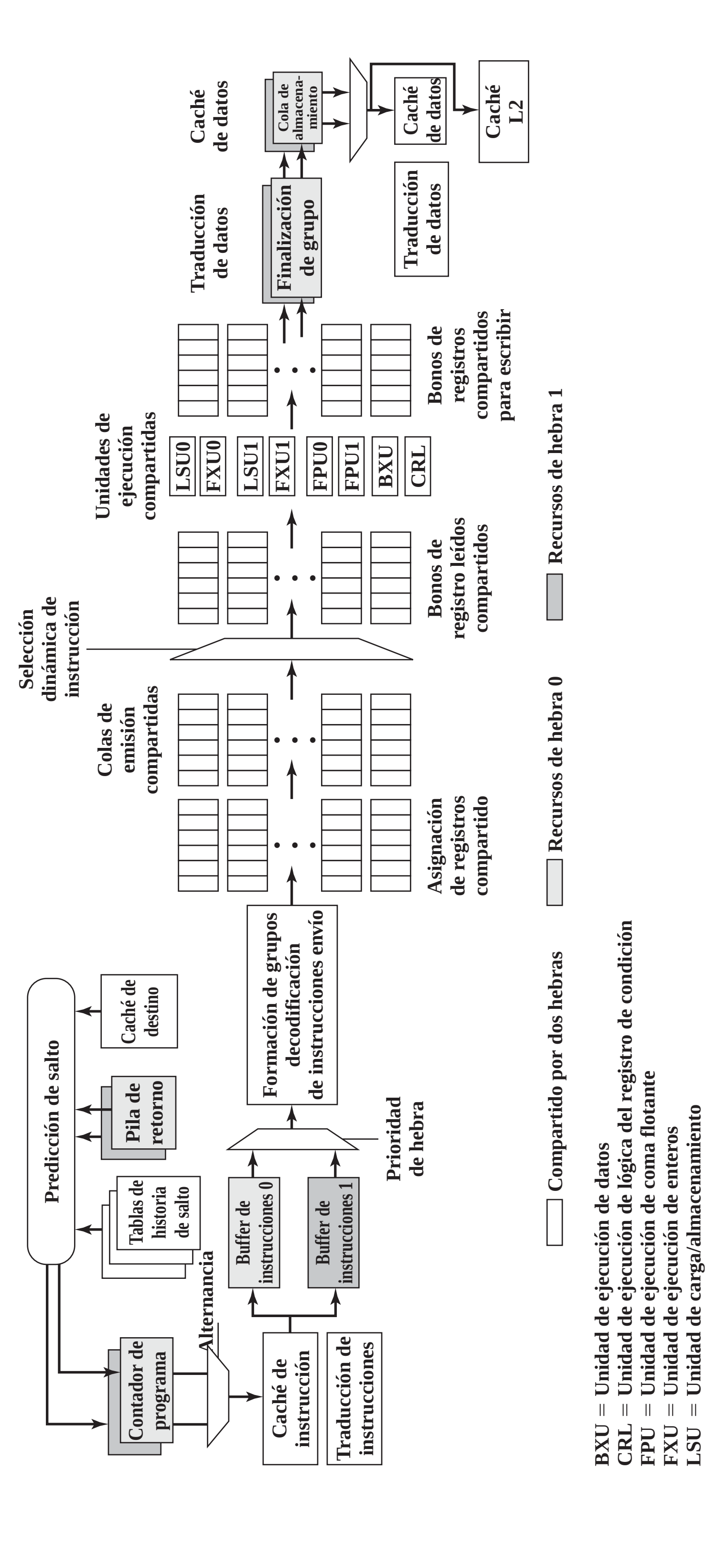


Figura 18.9. Flujo de las instrucciones en el Power5.

continuación, las instrucciones pasan a través de un recurso de renombramiento de registros en el mismo orden en el que se encuentran en el programa y se asignan registros lógicos a los registros físicos. El Power5 tiee 120 registros físicos de propósito general y 120 registros de coma flotante. Luego, las instrucciones pasan a las colas de emisión. Desde estas colas, las instrucciones se emiten utilizando multihebra simétrica. Es decir, el procesador tiene una arquitectura superescalar que puede emitir instrucciones de las dos hebras en paralelo. Al final del cauce, existen recursos separados para cada una de las hebras, necesarios para la finalización de las instrucciones.

18.5. *CLUSTERS*

Una de las áreas actualmente más activas es el diseño de *clusters*. Los *clusters* constituyen la alternativa a los multiprocesadores simétricos (SMP) para disponer de prestaciones y disponibilidad elevadas, y son particularmente atractivos en aplicaciones propias de un servidor.Se puede definir un *cluster* como un grupo de *computadores completos* interconectados que trabajan conjuntamente como un único recurso de cómputo, creándose la ilusión de que se trata de una sola máquina. El término *computador completo* hace referencia a un sistema que puede funcionar por sí solo, independientemente del *cluster.* Usualmente, en la literatura, cada computador del *cluster* se denomina nodo.

En [BREW97] se enumeran cuatro beneficios que pueden conseguirse con un *cluster*. Estos beneficios también pueden contemplarse como objetivos o requisitos de diseño:

- **Escalabilidad absoluta:** es posible configurar *clusters* grandes que incluso superan las prestaciones de los computadores independientes más potentes. Un cluster puede tener decenas de máquinas, cada una de las cuales puede ser un multiprocesador.
- **Escalabilidad incremental:** un *cluster* se configura de forma que sea posible añadir nuevos sistemas al *cluster* en ampliaciones sucesivas. Así, un usuario puede comenzar con un sistema modesto y ampliarlo a medida que lo necesite, sin tener que sustituir el sistema de que dispone por uno nuevo que proporcione mayores prestaciones.
- **Alta disponibilidad:** puesto que cada nodo del *cluster* es un computador autónomo, el fallo de uno de los nodos no significa la pérdida del servicio. En muchos casos, es el software el que proporciona automáticamente la tolerancia a fallos.
- **Mejor relación precio-prestaciones:** al utilizar elementos estandarizados, es posible es posible configurar un *cluster* con mayor o igual potencia de cómputo que un computador independiente mayor, a mucho menos costo.

CONFIGURACIONES DE *CLUSTERS*

En la literatura, los *clusters* se clasifican de formas muy diversas. Quizá la clasificación más sencilla es la que considera si los computadores comparten el acceso al mismo disco. La Figura 18.10a muestra un *cluster* de dos nodos en el que la interconexión se realiza mediante un enlace de alta velocidad que puede utilizarse para intercambiar mensajes que coordinan la actividad del *cluster*. El enlace puede ser una LAN que se comparte con otros computadores no incluidos en el *cluster,* o puede tratarse de un medio de interconexión específico. En este último caso, uno o varios computadores del

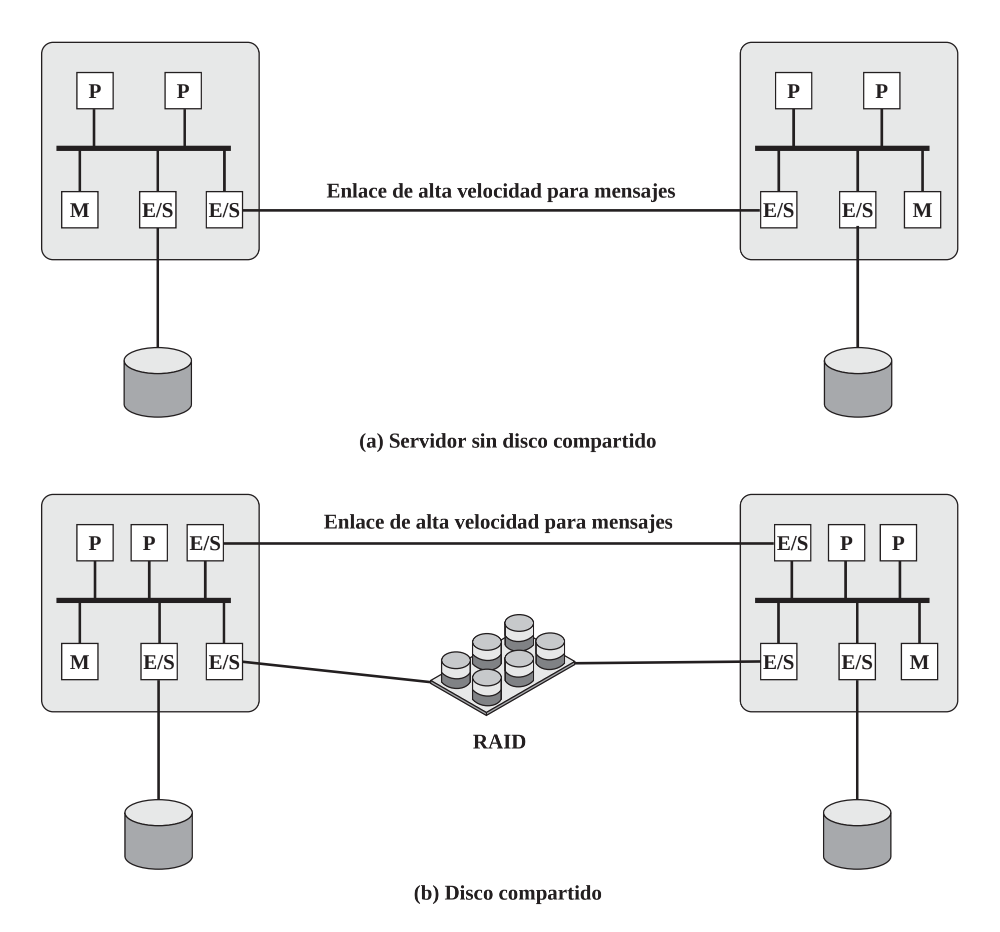


Figura 18.10. Configuraciones de *cluster.*

cluster tendrán un enlace a una LAN o a una WAN de forma que sea posible la conexión entre el *cluster*, actuando como servidor, y los clientes remotos. Considérese que en la figura cada computador se representa como multiprocesador. Esto no es imprescindible pero proporciona un aumento tanto de las prestaciones como de la disponibilidad.

En la clasificación simple de la Figura 18.10, la otra alternativa representada es un *cluster* con disco compartido. En este caso, generalmente también existe un enlace entre los nodos. Además, existe un subsistema de disco que se conecta directamente a los computadores del *cluster*. En la figura, el subsistema de disco común es un RAID. El uso del RAID o algún tipo de tecnología de discos redundantes es común en los *clusters* para que la elevada disponibilidad que se consigue con la presencia de varios computadores no se vea comprometida por un disco compartido que pueda convertirse en un único punto de fallo.

Se puede tener un panorama más claro del abanico de posibilidades de configuración de un *cluster* a partir de las alternativas funcionales. La Tabla 18.2 proporciona una clasificación muy útil a partir de las características de funcionamiento. Se discute a continuación.

Existe un procedimiento conocido como **espera pasiva** (*passive standby*), bastante antiguo y común, que simplemente consiste en mantener toda la carga de trabajo en un computador mientras

Tabla 18.2. Métodos de configuración de *clusters:* beneficios y limitaciones.

Método de configuración del *cluster*	**Descripción**	**Beneficios**	**Limitaciones**
Espera pasiva	Un servidor secundario sustituye al servidor primario en caso de que falle	Fácil de implementar	Alto coste debido a que el servidor secundario no está disponible para otras tareas de procesamiento
Secundario activo	El servidor secundario también se utiliza para tareas de procesamiento	Coste reducido porque los servidores secundarios pueden utilizarse para el procesamiento	Aumenta la complejidad
Servidores separados	Cada servidor tiene su propio disco. Los datos se copian desde el servidor primario al secundario	Alta disponibilidad	Penalización elevada en la red y el servidor debido a las operaciones de copia
Servidores conectados a discos	Los servidores se conectan a los mismos discos, pero cada servidor tiene sus propios discos. Si falla un servidor, otro servidor accede a sus discos.	Penalización reducida en la red y en el servidor debido a la eliminación de las operaciones de copia.	Usualmente se necesitan discos espejo o tecnología RAID para afrontar el riesgo de fallo de disco
Servidores compartiendo discos	Los servidores comparten simultáneamente el acceso a los discos	Penalización baja en la red y en el servidor. Riesgo reducido de fallo del sistema debido a un fallo de disco	Se necesita software de control de acceso exclusivo. Usualmente se utiliza con discos espejos o tecnología RAID

que otro permanece inactivo hasta tome el relevo del primero, cuando se produce un fallo de este. Para coordinar las máquinas, el computador activo, o primario, envía un «mensaje de actividad» (*heartbeat message*) al computador en espera. En el caso de que estos mensajes dejen de llegar, el computador en espera asume que el servidor ha fallado y se pone en marcha. Esta alternativa aumenta la disponibilidad pero no las prestaciones. Es más, si los dos computadores solo se intercambian el mensaje de actividad, y si no tienen discos comunes, entonces el computador en espera constituye un computador de reserva para ejecutar procesos pero no tiene acceso a las bases de datos gestionadas por el primer computador.

La espera activa no suele aplicarse en un *cluster*. El término *cluster* hace referencia a varios computadores interconectados que se encuentran activos realizando algún tipo de procesamiento, a la vez que proporcionan una imagen de sistema único al exterior. El término **secundario activo** (*active secondary*) se utiliza a menudo para referirse a esta configuración. Se pueden distinguir tres métodos para componer el *cluster*: con servidores separados (*separate server*), sin compartir nada (*shared nothing*), y compartiendo memoria (*shared memory*).

En una de las aproximaciones al diseño de *clusters*, cada computador es un **servidor independiente** con su propio disco y no existen discos compartidos por los sistemas (Figura 18.10a). Esta

organización proporciona tanto una disponibilidad como unas prestaciones elevadas. Se precisa algún tipo de software de gestión o planificación para asignar a los servidores las peticiones que se van recibiendo de los clientes, de forma que se equilibre la carga de los mismos y se consiga una utilización elevada. Es deseable tener una cierta capacidad de fallo, lo que significa que si un computador falla mientras está ejecutando una aplicación, otro computador del *cluster* puede acceder a la aplicación y completarla. Para que esto suceda, los datos se deben copiar constantemente en los distintos sistemas, de manera que cada procesador pueda acceder a los datos actuales de los demás. El coste que supone este intercambio de datos ocasiona una penalización en las prestaciones pero es el precio por conseguir una disponibilidad elevada.

Para reducir el coste que suponen las comunicaciones, la mayoría de los *clusters* están costituidos por servidores conectados a discos comunes (Figura 18.10b). Una variación a esta alternativa se designa como configuración **sin compartir nada** (*shared nothing*). En esta aproximación, los discos comunes se dividen en volúmenes diferentes y cada volumen pasa a pertenecer a un solo procesador. Si el computador falla, el *cluster* se debe reconfigurar de forma que los volúmenes que pertenecían al computador defectuoso pasen a los otros computadores.

También es posible hacer que los computadores compartan el disco al mismo tiempo (aproximación de **disco compartido**), para que todos los computadores tengan acceso a todos los volúmenes de todos los discos. Esta aproximación necesita utilizar algún tipo de procedimiento para el acceso exclusivo para asegurar que, en un momento dado, solo un computador puede acceder a los datos.

CONSIDERACIONES EN EL DISEÑO DEL SISTEMA OPERATIVO

Un completo aprovechamiento de la configuración hardware de un *cluster* exige una cierta ampliación de un sistema operativo propio de un único computador.

Gestión de los fallos. La forma en que se actúa frente a un fallo en un *cluster* depende del tipo de configuración del *cluster* (Tabla 18.2). En general, se pueden utilizar dos alternativas para enfrentarse a los fallos: *clusters* de alta disponibilidad y *clusters* tolerantes a fallos. Un *cluster* de alta disponibilidad es el que ofrece una probabilidad elevada de que todos sus recursos estén es servicio. Si se produce un fallo, tal como la caída del sistema o la pérdida de un volumen de disco, se pierden las tareas en curso. Si una de esas tareas se reinicia, puede ser un computador distinto el que le de servicio. No obstante, el sistema operativo del *cluster* no garantiza el estado de las transacciones ejecutadas parcialmente. Esto debería gestionarse en el nivel de aplicación.

Un *cluster* tolerante a fallos garantiza que todos los recursos estén disponibles. Esto se consigue utilizando discos compartidos redundantes y mecanismos para salvar las transacciones no terminadas y concluir las transacciones terminadas.

La función de conmutar aplicaciones y datos en el *cluster*, desde un sistema defectuoso a otro alternativo, se denomina *transferencia por fallo (failover)*. Una función adicional es la restauración de las aplicaciones y los datos por el sistema original, una vez superado el fallo; se denomina *recuperación después de un fallo (failback)*. La recuperación puede hacerse automáticamente, pero esto es deseable solo si el fallo ha sido completamente reparado y es poco probable que vuelva a producirse. En caso contrario, la recuperación automática puede ocasionar transferencias continuas en un sentido y en otro de programas y datos debido a la reaparición de un fallo, y se producirán problemas en cuanto a las prestaciones y a la recuperación.

Equilibrado de carga. Un *cluster* necesita una capacidad efectiva para equilibrar la carga entre los computadores deisponibles. Esta capacidad es necesaria para satisfacer el requisito de la escalabilidad incremental. Cuando un computador se añade al *cluster*, las aplicaciones encargadas de la asignación de tareas deberían incluir automáticamente a dicho computador junto con los restantes, para distribuir la carga de forma equilibrada. Los mecanismos de un nivel de software intermedio entre el sistema operativo y las aplicaciones *(middleware)* necesitan reconocer los servicios que pueden aparecer en los distintos miembros del *cluster* y pueden migrar desde un miembro a otro.

Computación paralela. En algunos casos, la utilización eficiente de un *cluster* requiere ejecutar en paralelo el software correspondiente a una aplicación. En [KAP00] se enumeran tres aproximaciones generales al problema:

- **Paralelización mediante el compilador:** un compilador con capacidad para generar código paralelo determina, en el momento de la compilación, las partes de la aplicación que pueden ejecutarse en paralelo. Estas se asignan a computadores distintos del *cluster*. Las prestaciones dependen de la naturaleza del problema y de lo bien que esté diseñado el compilador.
- **Paralelización realizada por el programador**: en este caso, el programador escribe la aplicación para que se ejecute en el *cluster*, y utiliza paso de mensajes para mover los datos entre los nodos del *cluster* según sea necesario. Esto aumenta la carga de trabajo del programador pero puede ser la mejor aproximación para sacar partido a un *cluster* en algunas aplicaciones.
- **Computación paramétrica**: esta aproximación puede utilizarse si la aplicación consiste básicamente en un algoritmo o programa que debe ejecutarse una gran cantidad de veces, pero cada vez con un conjunto diferente de condiciones de inicio o parámetros. Un buen ejemplo de esto es un modelo de simulación, que debe ejecutarse para un gran número de escenarios de forma que se obtengan resultados estadísticamente significativos. Para que esta aproximación sea efectiva se necesitan herramientas de procesamiento paramétrico que organicen, gestionen y envíen a ejecutar los trabajos de forma ordenada.

ARQUITECTURA DE LOS *CLUSTERS*

En la Figura 18.11 se muestra una arquitectura típica de *cluster*. Los computadores se conectan a través de una red de área local (LAN) de alta velocidad o mediante un conmutador. Cada computador puede trabajar de forma independiente. Además, en cada computador se instala una capa software intermedia (*middleware*) que permite el funcionamiento de todos los computadores como un único *cluster*. El *middleware* del *cluster* proporciona al usuario una imagen unificada conocida como **imagen de sistema único** (*single-system image*). El *middleware* también es responsable de proporcionar alta disponibilidad, distribuyendo la carga y respondiendo a los fallos de los componentes. En [HWAN99] se enumeran los servicios y funciones deseables en la capa de *middleware*:

- **Punto de entrada único**: un usuario entra al *cluster* igual que si entrara a un único computador.
- **Jerarquía de ficheros única**: el usuario ve una sola jerarquía de directorios de ficheros bajo el mismo directorio raíz.
- **Punto de control único**: por defecto, existe una única estación de trabajo que se utiliza para la gestión y el control del *cluster*.

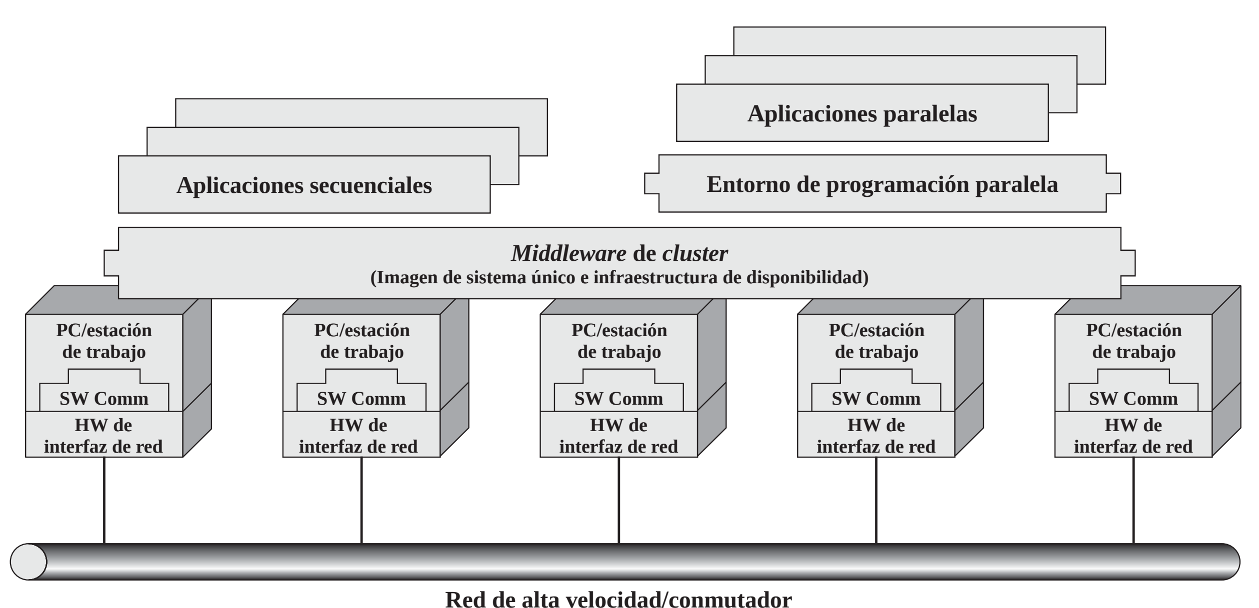


Figura 18.11. Arquitectura de computador de *cluster.*

- **Red virtual única**: un nodo puede acceder a cualquier otro nodo del *cluster*, incluso aunque la configuración presente en el cluster conste de varias redes interconectadas. Existe una única operación de red virtual.
- **Espacio de memoria único**: la memoria compartida distribuida permite que los programas compartan variables.
- **Sistema de gestión de trabajos único:** el planificador de trabajos de un *cluster* permite que un usuario pueda enviar un trabajo sin especificar el computador donde se va a ejecutar el trabajo.
- **Interfaz de usuario única:** existe una interfaz gráfica común para todos los usuarios, independientemente de la estación de trabajo desde la que se entre al *cluster.*
- **Espacio de E/S único:** cualquier nodo puede acceder remotamente a cualquier periférico de E/S o dispositivo de disco sin que se conozca su ubicación física.
- **Espacio de procesos único:** se utiliza un esquema de identificación de procesos uniforme. Un proceso de cualquier nodo puede crear o comunicarse con cualquier otro proceso en un nodo remoto.
- **Puntos de chequeo:** el estado del proceso y los resultados intermedios se almacenan periódicamente para permitir la recuperación después de un fallo.
- **Migración de procesos:** para permitir la distribución equilibrada de la carga.

Los cuatro últimos puntos de la lista anterior mejoran la disponibilidad del *cluster*. Los restantes contribuyen a la imagen de sistema único.

Volviendo a la Figura 18.11, un *cluster* también deberá incluir herramientas software para permitir una ejecución eficiente de los programas que pueden ejecutarse en paralelo.

CLUSTERS FRENTE A SISTEMAS SMP

Tanto los *clusters* como los multiprocesadores simétricos constituyen configuraciones con varios procesadores que pueden ejecutar aplicaciones con una alta demanda de recursos. Ambas soluciones están disponibles comercialmente, aunque los SMP lo están desde hace más tiempo.

La principal ventaja de un SMP es que resulta más fácil de gestionar y configurar que un *cluster*. El SMP está mucho más cerca del modelo de computador de un solo procesador para el que están disponibles casi todas las aplicaciones. El principal cambio que se necesita para pasar de un computador monoprocesador a un SMP se refiere al funcionamiento del planificador. Otra ventaja de un SMP es que necesita menos espacio físico y consume menos energía que un *cluster* comparable. Una última e importante ventaja es que los SMP son plataformas estables y bien establecidas.

Con el tiempo, no obstante, las ventajas de los *clusters* serán las que probablemente harán que sean estos los que dominen en el mercado de servidores de altas prestaciones. Los *clusters* son superiores a los SMP en términos de escalabilidad absoluta e incremental, y además también son superiores en términos de disponibilidad, puesto que todos los componentes del sistema pueden hacerse altamente redundantes.

18.6. ACCESO NO UNIFORME A MEMORIA

En términos comerciales, las dos alternativas para ejecutar aplicaciones en un sistema con varios procesadores son los SMP y los *clusters*. Durante algunos años, otra alternativa conocida como acceso no uniforme a memoria (NUMA, *Nonuniform memory access*) ha sido objeto de investigación y recientemente han aparecido computadores NUMA comerciales.

Antes de continuar, se deberían definir algunos términos que se encuentran a menudo en la literatura de computadores NUMA.

- **Acceso uniforme a memoria (UMA, *Uniform Memory Access*):** todos los procesadores pueden acceder a toda la memoria principal utilizando instrucciones de carga y almacenamiento. El tiempo de acceso de un procesador a cualquier región de la memoria es el mismo. El tiempo de acceso a memoria por parte de todos los procesadores es el mismo. La organización SMP discutida en las Secciones 18.2 y 18.3 es una organización UMA.
- **Acceso no uniforme a memoria (NUMA, *Nonuniform Memory Access*):** todos los procesadores tienen acceso a todas las partes de memoria principal utilizando instrucciones de carga y almacenamiento. El tiempo de acceso a memoria de un procesador depende de la región a la que se acceda. La última frase es cierta para todos los procesadores; no obstante, para procesadores distintos, las regiones de memoria que son más lentas o más rápidas son diferentes.
- **NUMA con coherencia de caché (CC-NUMA, *Cache-Coherent NUMA*):** un computador NUMA en el que la coherencia de caché se mantiene en todas las cachés de los distintos procesadores.

Un sistema NUMA sin coherencia de caché es más o menos equivalente a un *cluster*. Las alternativas que han despertado más interés comercial son los CC-NUMA, que son bastante diferentes de los SMP y los *clusters*. Usualmente, pero desafortunadamente no siempre, estos sistemas se denominan en la literatura comercial sistemas CC-NUMA. Esta sección se refiere únicamente a ellos.

MOTIVACIÓN

En un SMP existe un límite práctico en el número de procesadores que pueden utilizarse. Un esquema de caché eficaz reduce el tráfico en el bus entre los procesadores y la memoria principal. A medida que el número de procesadores se incrementa, el tráfico en el bus también aumenta. Además, el bus se utiliza para intercambiar señales de coherencia de caché, añadiendo más carga. A partir de cierto momento, el bus pasa a ser el cuello de botella para las prestaciones. La degradación de las prestaciones parece que limita el número de procesadores en una configuración SMP entre 16 y 64 procesadores. Por ejemplo el SMP Power Challenge de Silicon Graphics está limitado a 64 procesadores R10000 para un solo sistema, puesto que más allá de este número las prestaciones se degradan sustancialmente.

Precisamente el límite de procesadores en un SMP es uno de las motivos para el desarrollo de los *clusters*. Sin embargo, en un *cluster* cada nodo tienen su propia memoria principal privada y las aplicaciones no «ven» la memoria global. De hecho, la coherencia se mantiene mediante software en lugar de mediante hardware. Esta granularidad de memoria afecta a las prestaciones y, para conseguir el máximo nivel en ellas, el software debe ajustarse a este entorno. Una alternativa para conseguir multiprocesamiento a gran escala mientras se mantienen las características SMP son los computadores NUMA. Por ejemplo el NUMA Origin de Silicon Graphics está diseñado para incluir hasta 1 024 procesadores MIPS R10000 [WHIT97], y el computador NUMA-Q de Sequent se ha diseñado para soportar hasta 252 procesadores Pentium II [LOVE96].

El objetivo de un computador NUMA es mantener una memoria transparente desde cualquier parte del sistema, al tiempo que se permiten varios nodos de multiprocesador, cada uno con su propio bus u otro sistema de interconexión interna.

ORGANIZACIÓN

La Figura 18.12 muestra una organización CC-NUMA típica. Hay varios nodos independientes, cada uno de los cuales es, de hecho, un SMP. Así, cada nodo contiene varios procesadores, cada uno con sus cachés L1 y L2, más memoria principal. El nodo es el bloque básico de construcción de toda la organización CC-NUMA. Por ejemplo, cada nodo del Origin de Silicon Graphics incluye dos microprocesadores MIPS R10000; cada nodo del computador NUMA-Q de Sequent incluye cuatro procesadores Pentium II. Los nodos se interconectan a través de un medio de comunicación, que podría ser algún mecanismo de conmutación, un anillo o algún tipo de red.

Cada nodo de un sistema CC-NUMA incluye cierta cantidad de memoria principal. Sin embargo, desde el punto de vista de los procesadores, existe un único espacio de memoria direccionable en el que a cada posición se asocia una única dirección válida para todo el sistema. Cuando un procesador inicia un acceso a memoria, si la posición solicitada no se encuentra en la caché del procesador, entonces la caché L2 inicia una operación de captación. Si la línea deseada está en una posición remota de la memoria principal, la línea se capta a través del bus local. Si la línea solicitada está en una porción remota de la memoria, entonces se envía una petición automática para captar dicha línea a través de la red de interconexión, se proporciona a través del bus local a la caché que lo solicitaba en dicho bus. Toda esta actividad es automática y transparente al procesador y a su caché.

En esta configuración, la coherencia de caché es una cuestión central. Aunque las implementaciones pueden diferir en ciertos detalles, en términos generales podemos decir que cada nodo debe

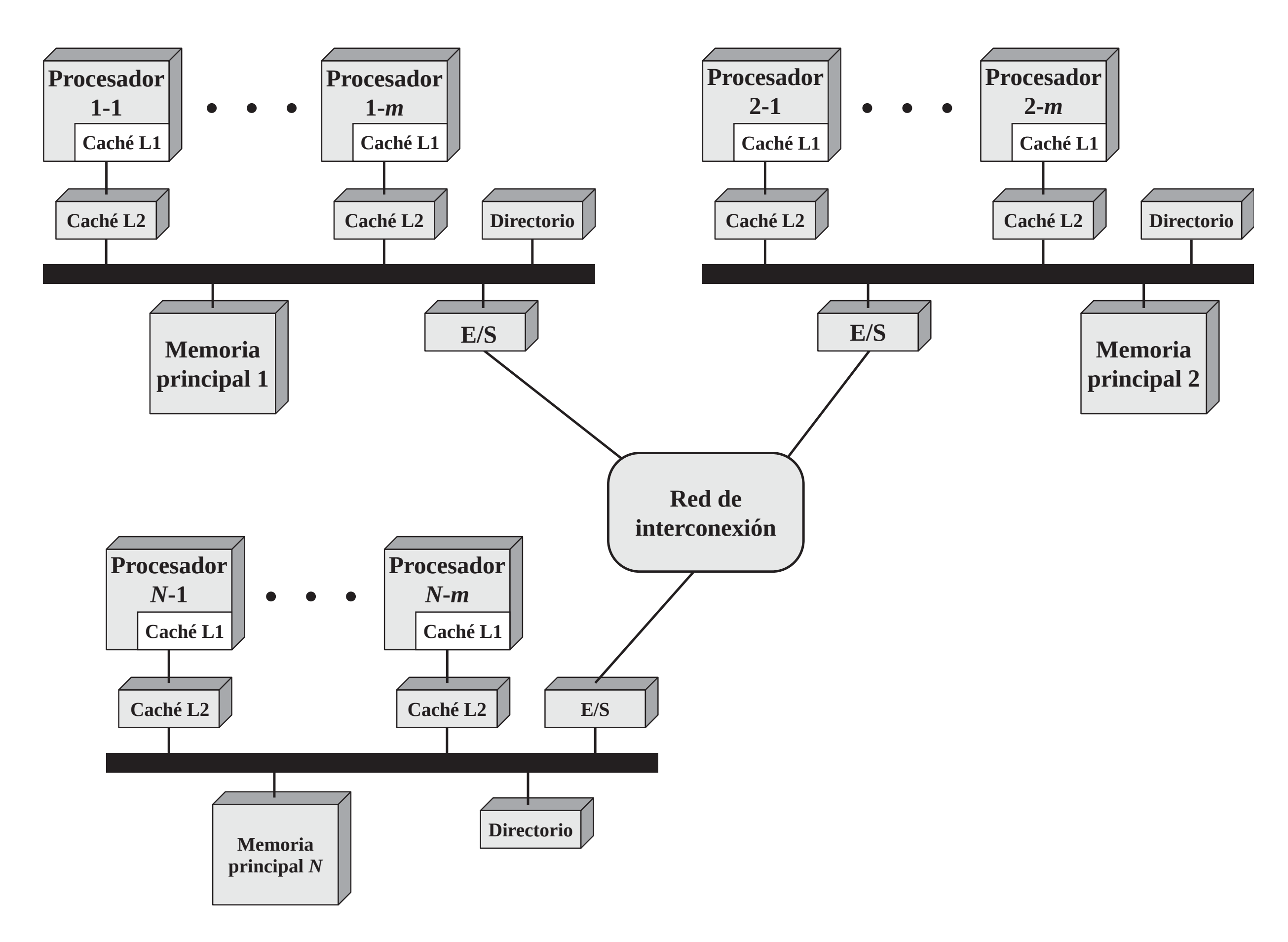


Figura 18.12. Organización CC-NUMA.

mantener algún tipo de orden de directorios que dé una indicación de la situación de varias partes de la memoria y también de la información de estado de la caché. Para ver cómo trabaja este esquema, utilizaremos un ejemplo tomado de [PFIS98]. Supóngase que el procesador 3 del nodo 2 (P2-3) solicita acceder a la posición 798, que está en la memoria del nodo 1. Se producirá la siguiente secuencia:

1. P2-3 genera la petición de lectura de la posición 798 a través del bus del nodo 2.
2. El módulo de direcciones del nodo 2 detecta la petición de acceso a memoria y determina que la dirección está en el módulo 1.
3. El módulo de direcciones del nodo 2 envía la petición al nodo 1, y esta es recogida por el módulo de direcciones del nodo 1.
4. El módulo de direcciones del nodo 1, actuando en sustitución del procesador P2-3, solicita la lectura de la posición 798.
5. La memoria principal del nodo 1 responde situando el dato solicitado en el bus.
6. El módulo de direcciones del nodo 1 toma el dato del bus.
7. El dato se transfiere al módulo de direcciones del nodo 2.

8. El módulo de direcciones del nodo 2 pone el dato en el bus de dicho nodo, actuando como si fuera la memoria que originariamente proporcionó el dato.
9. El dato se lee pasando a la caché del procesador P2-3 desde donde se proporciona a dicho procesador.

La secuencia precedente explica cómo se lee un dato desde una memoria remota utilizando mecanismos hardware que hacen la transacción transparente al procesador. Sobre estos mecanismos, se necesita algún tipo de protocolo de coherencia de caché. Los distintos sistemas se diferencian según la forma exacta en que se implemente ese protocolo. Aquí haremos únicamente algunas precisiones. En primer lugar, si en la secuencia previa el módulo de direcciones del nodo 1 guarda un registro indicando que alguna caché remota tiene una copia de la línea que contiene a la posición 798, se necesita un protocolo cooperativo que tenga en cuenta las posibles modificaciones. Por ejemplo, si se hace una modificación en la caché, este hecho debe comunicarse al resto de nodos. En el módulo de direcciones de cada nodo que recibe la notificación de una modificación se puede determinar si hay alguna caché local que tenga esa línea, y si es así hace que la misma se invalide. Si la posición de memoria considerada se encuentra en el nodo que recibe la notificación de modificación, el módulo de direcciones necesita mantener un registro que indique que la línea de memoria en cuestión se ha invalidado y permanece en este estado hasta que sea sustituida. Si otro procesador (local o remoto) solicita la línea invalidada, entonces el módulo de direcciones local debe forzar una escritura que actualice el dato en memoria antes de proporcionar el dato.

PROS Y CONTRAS DE UN COMPUTADOR NUMA

La principal ventaja de un computador CC-NUMA es que puede proporcionar un nivel de prestaciones efectivo con mayores niveles de paralelismo que un SMP, sin que se precisen cambios importantes en el software. Con varios nodos NUMA, el tráfico del bus en cualquier nodo se limita a las peticiones que el bus puede manejar. No obstante, si muchos de los accesos a memoria se producen a nodos remotos, las prestaciones empiezan a reducirse. Existe una razón para creer que esta reducción de prestaciones puede evitarse. En primer lugar, el uso de las cachés L1 y L2 permite reducir los accesos a memoria, incluyendo los remotos. Si la mayoría del software tiene una localidad temporal grande, los accesos a memorias remotas no deberían ser muchos. Segundo, si el software tiene una localidad espacial buena y se utiliza memoria virtual, los datos que se necesitan en la aplicación residirán en un número limitado de páginas frecuentemente usadas, que pueden cargarse en la memoria local de la aplicación que se está ejecutando. Un informe de los diseñadores de Sequent indica que esta localidad espacial aparece en ciertas aplicaciones representativas [LOVE96]. Finalmente, el esquema de memoria virtual puede mejorarse incluyendo en el sistema operativo un mecanismo de migración de páginas que mueva una página de memoria virtual al nodo que la utiliza frecuentemente. Un informe de los diseñadores de Silicon Graphics indica que han tenido éxito con esta aproximación [WHIT97].

La alternativa CC-NUMA también tiene desventajas. Dos de ellas se discuten con detalle en [PFIS98]. En primer lugar, un computador CC-NUMA no parece tan transparente como un SMP; se necesitan ciertos cambios en el software para adaptar el sistema operativo y las aplicaciones desde un SMP a un CC-NUMA. Entre los cambios en el sistema operativo están la ya mencionada asignación de páginas, la asignación de procesos y el equilibrado de la carga. Un segundo aspecto a considerar es la disponibilidad. Se trata de una cuestión bastante compleja que depende de la implementación exacta del CC-NUMA. Los lectores interesados pueden consultar [PFIS98].

18.7. COMPUTACIÓN VECTORIAL

Aunque las prestaciones de los grandes computadores (*mainframes*) de propósito general continúa aumentando sin tregua, siguen existiendo aplicaciones que están fuera del alcance de los *mainframes* actuales. Se necesitan computadores que resuelvan problemas matemáticos de procesos reales, tales como los que aparecen en disciplinas como la aerodinámica, sismología, meteorología y física atómica, nuclear y de plasmas.

Típicamente, estos problemas se caracterizan por necesitar una precisión elevada y programas que realicen de forma repetitiva operaciones aritméticas en coma flotante con grandes matrices de números. La mayoría de estos problemas pertenecen a la categoría conocida como *simulación de espacios continuos (continuous-field simulation)*. En esencia, una situación física (por ejemplo, el flujo de aire próximo a la superficie de un cohete) se puede describir mediante una región de tres dimensiones. Esta región se aproxima mediante una retícula de puntos. Un conjunto de ecuaciones diferenciales definen el comportamiento físico de la superficie en cada punto. Esas ecuaciones se representan como una matriz de valores y coeficientes, y su resolución implica repetidas operaciones aritméticas sobre las matrices de datos.

Para manejar este tipo de problemas, se han desarrollado supercomputadores. Típicamente, estas máquinas son capaces de realizar cientos de millones de operaciones en coma flotante por segundo, y cuestan entre diez y quince millones de dólares. A diferencia de los *mainframes*, que se diseñan para la multiprogramación y las E/S intensivas, los supercomputadores están optimizados para el tipo de cálculo numérico descrito.

El supercomputador tiene un uso limitado y, debido a su precio, un mercado limitado. Comparativamente, pocas máquinas de este tipo están operativas, en su mayoría en centros de investigación y en algunas agencias gubernamentales con actividades científicas o de ingeniería. Igual que en otras áreas de la tecnología de computadores, hay una constante demanda para mejorar las prestaciones de los supercomputadores. En consecuencia, la tecnología y las prestaciones de los supercomputadores continúan evolucionando.

Hay otro tipo de sistemas que han sido diseñados para las necesidades de la computación vectorial: se trata de los *procesadores matriciales*. Aunque un supercomputador está optimizado para la computación vectorial, es un computador de propósito general, capaz de procesar escalares y realizar tareas generales de procesamiento de datos. Los procesadores matriciales no realizan el procesamiento escalar; están configurados como dispositivos periféricos para que los usuarios de grandes computadores y minicomputadores puedan ejecutar partes vectorizadas de sus programas.

APROXIMACIONES A LA COMPUTACIÓN VECTORIAL

La clave para el diseño de un supercomputador o un procesador matricial es reconocer que la tarea principal consiste en realizar operaciones sobre matrices o vectores de números en coma flotante. En un computador de propósito general, esto precisaría realizar una iteración para cada elemento de la matriz. Por ejemplo, considérense dos vectores (matriz unidimensional) de números, A y B. Se desea sumarlos y situar el resultado en C. En el ejemplo de la Figura 18.13, esto precisaría seis sumas. ¿Cómo podríamos acelerar este cálculo? La respuesta está en aprovechar alguna forma de paralelismo.

$$\begin{bmatrix} 1.5 \\ 7.1 \\ 6.9 \\ 100.5 \\ 0 \\ 59.7 \end{bmatrix} + \begin{bmatrix} 2.0 \\ 39.7 \\ 1000.003 \\ 11 \\ 21.1 \\ 19.7 \end{bmatrix} = \begin{bmatrix} 3.5 \\ 46.8 \\ 1006.093 \\ 111.5 \\ 21.1 \\ 79.4 \end{bmatrix}$$

$$A \quad + \quad B \quad = \quad C$$

Figura 18.13. Ejemplo de suma vectorial.

Se han seguido diversas aproximaciones para lograr paralelismo en la computación vectorial. Lo ilustramos con un ejemplo. Considérese la multiplicación

$C = A \times B$, donde A, B y C son matrices de $N \times N$. La fórmula para cada elemento de C es

$$c_{i,j} = \sum_{k=1}^{N} a_{i,k} \times b_{k,j}$$

donde los elementos de A, B y C son $a_{i,j}$, $b_{i,j}$, y $c_{i,j}$, respectivamente. La Figura 18.14a muestra un programa FORTRAN para este cálculo que puede ejecutarse en un procesador escalar ordinario.

Una posibilidad para mejorar las prestaciones se denomina *procesamiento vectorial*. Este asume que es posible operar sobre un vector de datos unidimensional. La Figura 18.14b es un programa FORTRAN con una nueva forma de instrucción que permite especificar cálculos vectoriales. La notación (J = 1, N) indica que las operaciones con los índices J del intervalo dado se realizan como una sola operación. La forma de conseguir esto se tratará en breve.

El programa de la Figura 18.14b indica que todos los elementos de la fila i-ésima se calculan en paralelo. Cada elemento de la fila es una suma, y las sumas (sobre K) se hacen en serie, no en paralelo. Incluso así, solo se necesitan N^2 multiplicaciones vectoriales para este algoritmo en comparación con las N^3 multiplicaciones escalares del algoritmo escalar.

Otro enfoque, el *procesamiento paralelo*, se ilustra en la Figura 18.14c. Esta aproximación asume que se dispone de N procesadores independientes que pueden funcionar en paralelo. Para utilizar los procesadores eficazmente, se deben repartir los cálculos de alguna forma entre los procesadores. Se utilizan dos primitivas. La primitiva FORK n hace que comience un proceso independiente en la posición n. Entretanto, el proceso original continúa la ejecución en la instrucción que sigue al FORK. Cada ejecución de un FORK genera un nuevo proceso. La instrucción JOIN es esencialmente la inversa al FORK. La sentencia JOIN N hace que N procesos independientes se fundan en uno que continúa la ejecución en la instrucción que sigue al JOIN. El sistema operativo debe coordinar esta fusión, de forma que la ejecución no continúe hasta que los N procesos hayan alcanzado la instrucción JOIN.

El programa de la Figura 18.14c está escrito imitando al comportamiento del programa de procesamiento vectorial. En el programa de procesamiento paralelo, cada columna de C es calculada por un proceso distinto. Así, los elementos de una fila de C se calculan en paralelo.

La discusión precedente describe las aproximaciones a la computación vectorial en términos lógicos o de la arquitectura. Volvamos ahora a considerar las formas de organización de los procesadores

```
        DO 100 I = 1, N
        DO 100 J = 1, N
        C(I, J) = 0.0
        DO 100 K = 1, N
        C(I, J) = C(I, J) + A(I, K) + B(K, J)
100     CONTINUE
```

(a) Procesamiento escalar

```
        DO 100 I = 1, N
        C(I, J) = 0.0 (J = 1, N)
        DO 100 K = 1, N
        C(I, J) = C(I, J) + A(I, K) + B(K, J) (J = 1, N)
100     CONTINUE
```

(b) Procesamiento vectorial

```
        DO 50 J = 1, N − 1
        FORK 100
50      CONTINUE
        J = N
100     DO 200 I = 1, N
        C(I, J) = 0.0
        DO 200 K = 1, N
        C(I, J) = C(I, J) + A(I, K) + B(K, J)
200     CONTINUE
```

(c) Procesamiento paralelo

Figura 18.14. Multiplicación de matrices ($C = A \times B$).

que pueden utilizarse para implementar estas aproximaciones. Una amplia variedad de organizaciones han sido y están siendo consideradas. Las tres categorías principales son:

- ALU segmentada
- ALU paralelas
- Procesadores paralelos

La Figura 18.15 muestra las primeras dos aproximaciones. La segmentación de cauce ya se discutió en el Capítulo 12. Aquí se extiende el concepto a la operación de la ALU. Puesto que las operaciones en coma flotante son bastante complejas, existe la posibilidad de descomponer una operación en coma flotante en etapas, de manera que las diferentes etapas puedan operar concurrentemente sobre conjuntos de datos distintos. Esto se ilustra en la Figura 18.16a. La suma en coma flotante se divide en cuatro etapas (*véase* la Figura 9.22): comparación, desplazamiento, suma y normalización. Un vector de números se introduce secuencialmente a través de la primera etapa. A medida que prosigue el procesamiento, se operarán concurrentemente en el cauce cuatro conjuntos de números diferentes.

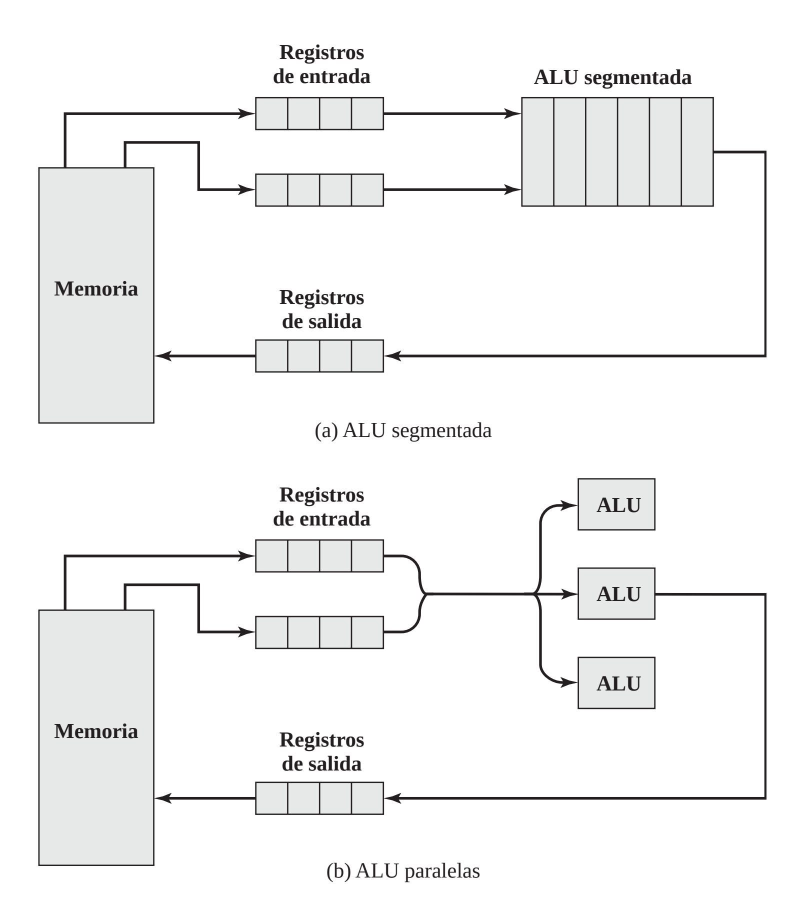


Figura 18.15. Alternativas para la computación vectorial.

Debe quedar claro que esta organización es adecuada para el procesamiento vectorial. Para verlo, considérese el cauce de instrucciones descrito en el Capítulo 12. El procesador ejecuta un ciclo repetitivo de captación y procesamiento de instrucciones. En ausencia de saltos, el procesador está contínuamente captando instrucciones de posiciones consecutivas. Por tanto, el cauce se mantiene lleno y se consigue una reducción en el tiempo de ejecución. Igualmente, una ALU segmentada ganará tiempo solo si se alimenta con una secuencia de datos de posiciones consecutivas. La ejecución de una única operación en coma flotante aislada no se acelera con un cauce. El aumento de velocidad se consigue cuando se presenta a la ALU un vector de operandos. La unidad de control introduce ciclo a ciclo los datos en la ALU hasta que se ha procesado el vector completo.

La operación del cauce se puede mejorar si los elementos del vector están disponibles en registros en lugar de en memoria principal. Este hecho se sugiere en la Figura 18.15a. Los elementos de cada operando vectorial se cargan como un bloque en un registro vectorial, que es simplemente un banco grande de registros idénticos. El resultado también se sitúa en un registro vectorial. Así, la mayoría de las operaciones implican solo el uso de registros, y solo las operaciones de carga y almacenamiento, y el comienzo y el final de una operación vectorial, necesitan acceder a memoria.

Los mecanismos que se ilustran en la Figura 18.16 se podrían denominar *segmentación de cauce dentro de una operación*. Es decir, tenemos una única operación aritmética (por ejemplo, $C = A + B$)

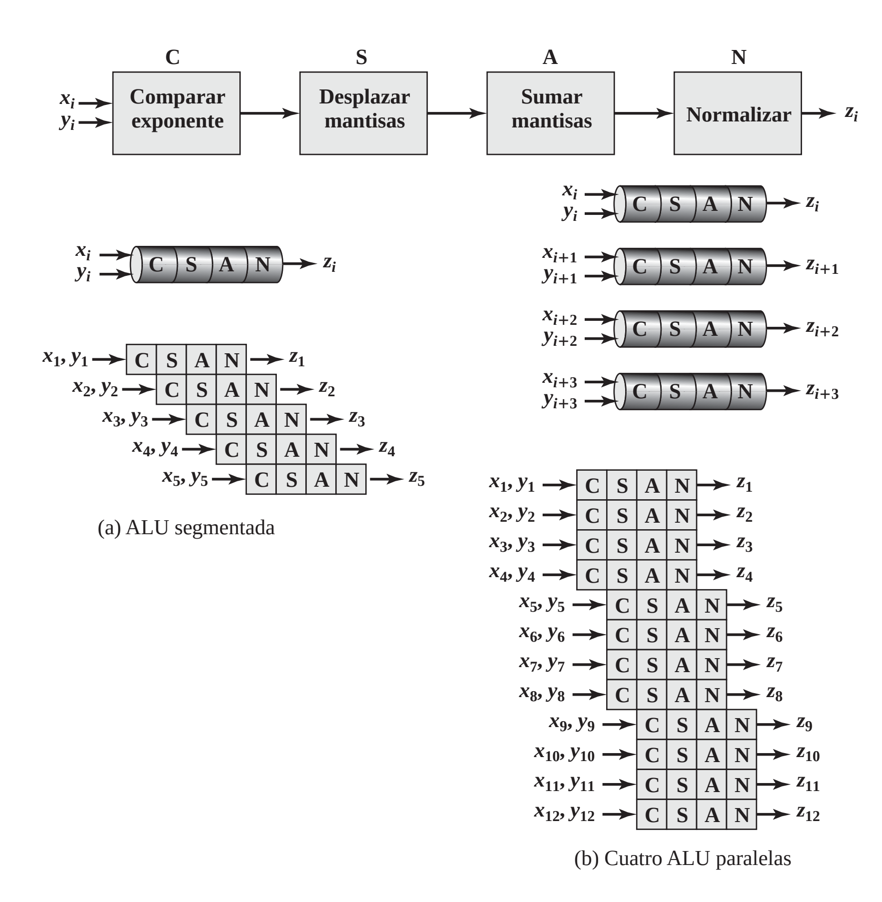


Figura 18.16. Procesamiento segmentado.

que se aplica a operandos vectoriales, y la segmentación de cauce permite que múltiples elementos vectoriales se procesen en paralelo. Este mecanismo se puede completar con la *segmentación de cauce de operaciones*. En este caso, hay una secuencia de operaciones aritméticas vectoriales, y se utiliza la segmentación de cauce de instrucciones para acelerar el procesamiento. Una aproximación a esto, conocida como **encadenamiento** (*chaining*), aparece en los supercomputadores Cray. La regla básica para el encadenamiento es esta: una operación vectorial puede empezar tan pronto como el primer elemento del (de los) operando(s) vectorial(es) está disponible y la unidad funcional (es decir, suma, resta, multiplicación, división) está libre. En esencia, el encadenamiento hace que los resultados generados por una unidad funcional pasen inmediatamente a alimentar otra unidad funcional, y así sucesivamente. Si se utilizan registros vectoriales, los resultados intermedios no tienen que almacenarse en memoria y pueden utilizarse incluso antes de que la operación vectorial que los origina haya terminado por completo.

Por ejemplo, cuando se calcula $C = (s \times A) + B$, donde A, B, y C son vectores y s es un escalar, el Cray puede ejecutar tres instrucciones de una vez. Los elementos captados por una instrucción de carga (*Load*) entran inmediatamente al multiplicador encauzado, los productos se envían al sumador encauzado y las sumas se sitúan en un registro vectorial tan pronto como el sumador las realiza:

1. Carga Vectorial $\quad$ A → Registro Vectorial (VR1)
2. Carga Vectorial $\quad$ B → VR2

3. Multiplicación Vectorial   s × VR1 → VR3
4. Suma Vectorial   VR3 + VR2 → VR4
5. Almacenamiento Vectorial   VR4 → C

Las instrucciones 2 y 3 pueden encadenarse (encauzadas) puesto que implican posiciones de memoria y registros distintos. La instrucción 4 necesita los resultados de las instrucciones 2 y 3, pero también puede encadenarse con ellas. Tan pronto como los primeros elementos del registo vectorial 2 y 3 estén disponibles, puede empezar la operación de la instrucción 4.

Otra forma de conseguir el procesamiento vectorial es con el uso de varias ALU en un solo procesador, bajo el control de una única unidad de control. En este caso, la unidad de control enruta los datos hacia las ALU para que puedan funcionar en paralelo. Es también posible utilizar segmentación de cauce en cada una de las ALU paralelas. Esto se ilustra en la Figura 18.16b. El ejemplo muestra un caso en el que cuatro ALU operan en paralelo.

Como la organización de segmentación de cauce, la organización de ALU paralelas se ajusta bien al procesamiento vectorial. La unidad de control introduce los elementos vectoriales en las ALU de forma cíclica hasta que se han procesado todos los elementos. Esta organización es más compleja que una única ALU de un CPI (Ciclo Por Instrucción).

Finalmente, el procesamiento vectorial puede conseguirse utilizando varios procesadores paralelos. En este caso, es necesario dividir la tarea en múltiples procesos que se ejecutan en paralelo. Esta organización es eficiente solo si se dispone de software y hardware para la coordinación efectiva de los procesadores paralelos. Podemos expandir nuestra taxonomía de la Sección 18.1 para reflejar estas nuevas estructuras, tal y como muestra la Figura 18.17. Las organizaciones de computador pueden distinguirse por la presencia de una o más unidades de control. Múltiples unidades de control implican múltiples procesadores. Siguiendo nuestra discusión previa, si los múltiples procesadores pueden cooperar en la ejecución de una tarea dada, se denominan *procesadores paralelos*.

El lector debe tener cuidado con alguna terminología desafortunada que probablemente encuentre en la literatura. El término *procesador vectorial* se utiliza a menudo como equivalente a organización de ALU segmentada, aunque una organización de unidades ALU paralelas también está diseñada para el procesamiento vectorial y, como hemos indicado, una organización de procesadores paralelos también puede diseñarse para el procesamiento vectorial. El término *procesador matricial* se utiliza a veces para referirse a las ALU paralelas aunque, nuevamente, cualquiera de las tres organizaciones está optimizada para el procesamiento de matrices. Para empeorar las cosas, el término *procesador matricial* usualmente hace referencia a un procesador auxiliar conectado a un procesador de propósito general y utilizado para realizar cálculos vectoriales. Un procesador matricial puede utilizar tanto la aproximación de ALU segmentada o ALU paralela.

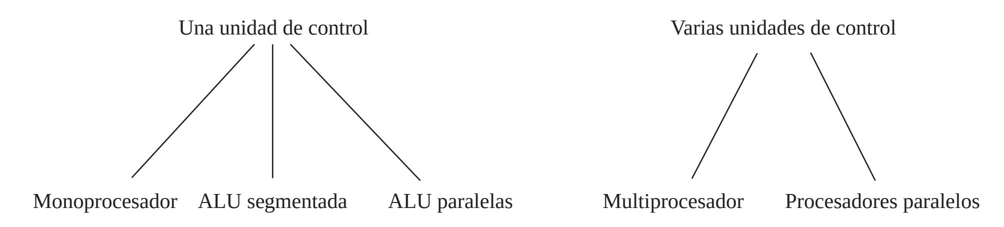


Figura 18.17. Taxonomía de organizaciones de computador.

Actualmente, la organización de ALU segmentada domina el mercado. Los sistemas con segmentación de cauce son menos complejos que otras aproximaciones. El diseño de sus unidades de control y sistemas operativos está lo suficientemente desarrollado para conseguir una asignación de recursos eficiente y elevadas prestaciones. El resto de esta sección se dedica a un examen más detallado de esta aproximación, utilizando un ejemplo específico.

UNIDAD VECTORIAL IBM 3090

Un buen ejemplo de una organización con ALU segmentada para el procesamiento vectorial es la unidad vectorial desarrollada para la arquitectura IBM 370 e implementada en la serie 3090 de la gama alta [PADE88, TUCK87]. Esta unidad constituye una extensión opcional al sistema básico pero está altamente integrada en él. Recuerda a los recursos vectoriales que se encuentran en supercomputadores, tales como los de la familia Cray.

La unidad IBM utiliza una serie de registros vectoriales. Cada registro es en realidad un banco de registros escalares. Para calcular la suma $C = A + B$, los vectores A y B se cargan en dos registros vectoriales. Desde estos registros, los datos pasan tan rápido como es posible a través de la ALU, y los resultados se almacenan en un tercer registro vectorial. El solapamiento de los cálculos, y la carga en bloque de los datos de entrada en los registros, producen un significativo aumento de la velocidad respecto a la operación en una ALU ordinaria.

Organización. La arquitectura vectorial de IBM y de ALU segmentadas vectoriales similares, proporciona mejoras en las prestaciones respecto a los bucles de instrucciones aritméticas escalares de tres maneras:

- La estructura fija y predeterminada de los datos vectoriales permite reemplazar las instrucciones incluidas en bucles por rápidas operaciones máquina internas (cableadas o microprogramadas).
- El acceso a los datos y las operaciones aritméticas con elementos vectoriales sucesivos se pueden completar concurrentemente solapando las operaciones en un cauce segmentado o realizando en paralelo operaciones sobre múltiples elementos.
- El uso de registros vectoriales para los resultados intermedios evita referencias adicionales a memoria.

La Figura 18.18 muestra la organización general de la unidad vectorial. Aunque la unidad vectorial aparece como un elemento físicamente distinto que se añade al procesador, su arquitectura es una extensión de la arquitectura S/370 y es compatible con ella. La unidad vectorial está integrada en la arquitectura S/370 de la siguiente forma:

- Las instrucciones existentes en la S/370 se utilizan para todas las operaciones escalares.
- Las operaciones aritméticas sobre elementos individuales de un vector producen exactamente el mismo resultado que las instrucciones escalares del S/370. Por ejemplo, esta decisión de diseño concierne a la definición del resultado de una operación DIVIDE en coma flotante. ¿El resultado debe ser exacto, como en la división en coma flotante escalar, o bien se admite una aproximación que permita una implementación de mayor velocidad pero que en ciertas ocasiones podría producir un error en uno o en más bits de las posiciones menos significativas?

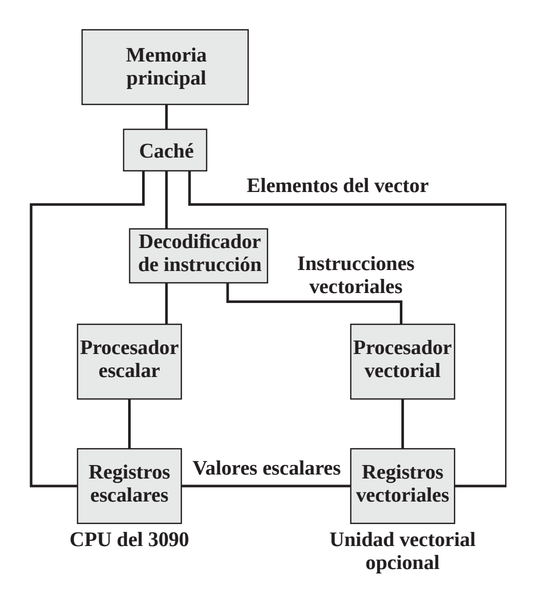


Figura 18.18. IBM 3090 con unidad vectorial.

La decisión tomada permite una compatibilidad ascendente completa con el S/370 a expensas de una pequeña reducción en las prestaciones.

- Las instrucciones vectoriales se pueden interrumpir, y su ejecución puede continuarse desde el punto de interrupción después de realizar la acción apropiada, de forma compatible con el esquema de interrupción de programas en el S/370.
- Las excepciones aritméticas son las mismas, o extensiones de las mismas excepciones propias de las instrucciones aritméticas escalares del S/370, y se utilizan rutinas similares para su tratamiento. Para esto, se emplea un índice de interrupción vectorial que indica la posición del registro vectorial afectado por la excepción (por ejemplo, desbordamiento u *overflow*). Así, cuando se reanuda la ejecución de la instrucción vectorial, se accede a la posición adecuada del registro vectorial.
- Los datos vectoriales residen en la memoria virtual, manejándose las faltas de página en la forma estándar.

Este nivel de integración proporciona una serie de beneficios. Los sistemas operativos existentes pueden dar cobertura a la unidad vectorial con pequeñas extensiones. Los programas de aplicación, compiladores y demás software existente puede ejecutarse sin cambios. El software que pueda beneficiarse de la unidad vectorial puede modificarse según convenga.

Registros. Un aspecto clave del diseño de la unidad vectorial es si los operandos se sitúan en registros o en memoria. La organización IBM se denomina *registro-a-registro* porque los operandos, tanto los de entrada como los de salida, pueden encontrarse en registros vectoriales. Esta aproximación también se utiliza en el supercomputador Cray. Una alternativa, utilizada en las máquinas de Control Data, consiste en obtener los operandos directamente desde la memoria. La principal desventaja del uso de registros vectoriales es que el programador, o el compilador, debe tenerlo en cuenta. Por ejemplo, supóngase que la longitud de los registros vectoriales es K y las longitudes de los

vectores procesados es $N > K$. En este caso, debe introducirse un bucle vectorial, en el que la operación se realiza sobre K elementos cada vez y el bucle se repite N/K veces. La principal ventaja de la aproximación de registros vectoriales es que la operación se desacopla de la memoria, más lenta, y en cambio utiliza principalmente registros.

El incremento de velocidad que se puede conseguir al utilizar registros se muestra en la Figura 18.19. La rutina FORTRAN multiplica el vector A por el vector B para producir el vector C, donde cada vector tiene una parte real (AR,BR,CR) y una imaginaria (AI,BI,CI). El 3090 puede realizar un ciclo de acceso (lectura o escritura) a memoria principal por ciclo de procesador, o de reloj, tiene registros que permiten mantener dos accesos para lectura y uno para escritura por ciclo, y produce un resultado por ciclo de su unidad aritmética. Asumamos que se utilizan instrucciones que pueden especificar dos operandos fuente y un resultado[4]. Una parte de la figura muestra que, con instrucciones de memoria-a-memoria, cada iteración del cálculo necesita un total de 18 ciclos. Con

RUTINA FORTRAN:

```
      DO 100 J = 1, 50
      CR(J) = AR(J) * BR(J) − AI(J) * BI(J)
100   CI(J) = AR(J) * BI(J) + AI(J) * BR(J)
```

Operación	Ciclos
AR(J) * BR(J) → T1(J)	3
AI(J) * BI(J) → T2(J)	3
T1(J) − T2(J) → CR(J)	3
AR(J) * BI(J) → T3(J)	3
AI(J) * BR(J) → T4(J)	3
T3(J) + T4(J) → CI(J)	3
TOTAL	18

(a) Memoria a memoria

Operación	Ciclos
AR(J) → V1(J)	1
BR(J) → V2(J)	1
V1(J) * V2(J) → V3(J)	1
AI(J) → V4(J)	1
BI(J) → V5(J)	1
V4(J) * V5(J) → V6(J)	1
V3(J) − V6(J) → V7(J)	1
V7(J) → CR(J)	1
V1(J) * V5(J) → V8(J)	1
V4(J) * V2(J) → V9(J)	1
V8(J) + V9(J) → V0(J)	1
V0(J) → CI(J)	1
TOTAL	12

(b) Registro a registro

Operación	Ciclos
AR(J) → V1(J)	1
V1(J) * BR(J) → V2(J)	1
AI(J) → V3(J)	1
V3(J) * BI(J) → V4(J)	1
V2(J) − V4(J) → V5(J)	1
V5(J) → CR(J)	1
V1(J) * BI(J) → V6(J)	1
V4(J) * BR(J) → V7(J)	1
V6(J) + V7(J) → V8(J)	1
V8(J) → CI(J)	1
TOTAL	10

(c) Memoria a registro

Operación	Ciclos
AR(J) → V1(J)	1
V1(J) * BR(J) → V2(J)	1
AI(J) → V3(J)	1
V2(J) − V3(J) * BI(J) → V2(J)	1
V2(J) → CR(J)	1
V1(J) * BI(J) → V4(J)	1
V4(J) + V3(J) * BR(J) → V5(J)	1
V5(J) → CI(J)	1
TOTAL	8

(d) Instrucciones compuestas

Vi = Registros vectoriales
AR, BR, AI, BI = Operandos en memoria
Ti = Posiciones temporales en memoria

Figura 18.19. Programas alternativos de computación vectorial.

[4] Para la arquitectura 370, las únicas instrucciones de tres operandos (instrucciones con registro y memoria, RS, *Register and Store*) especifican dos operandos en registros y uno en memoria. En la parte a de este ejemplo, asumimos la existencia de operaciones de tres operandos en las que todos los operandos están en memoria. Esto se hace con la intención de realizar la comparación y, de hecho, se podría haber elegido ese formato de instrucción para la arquitectura vectorial.

una arquitectura pura de registro-a-registro (parte b), este tiempo se reduce a doce ciclos. Por supuesto, con la operación de registro-a-registro, las cantidades vectoriales deben cargarse en los registros vectoriales antes del cálculo y almacenarse en memoria después. Para vectores largos, esta penalización ocasionada es relativamente pequeña. La Figura 18.19c muestra que la posibilidad de especificar en una instrucción tanto datos en memoria como en registros vectoriales, reduce el tiempo a diez ciclos por iteración. Este último tipo de instrucción está incluido en la arquitectura vectorial[5].

La Figura 18.20 ilustra los registros que forman parte de la unidad vectorial IBM 3090. Hay 16 registros vectoriales de 32 bits. Los registros vectoriales también pueden acoplarse para constituir ocho registros vectoriales de 64 bits. Cualquier elemento del registro puede almacenar un valor entero o en coma flotante. Así pues, los registros vectoriales pueden utilizarse para valores enteros de 32 y 64 bits, y para valores en coma flotante de 32 y 64 bits.

La arquitectura especifica que cada registro contiene entre ocho y 512 elementos escalares. La selección de la longitud en cada caso significa un compromiso de diseño. El tiempo para realizar una

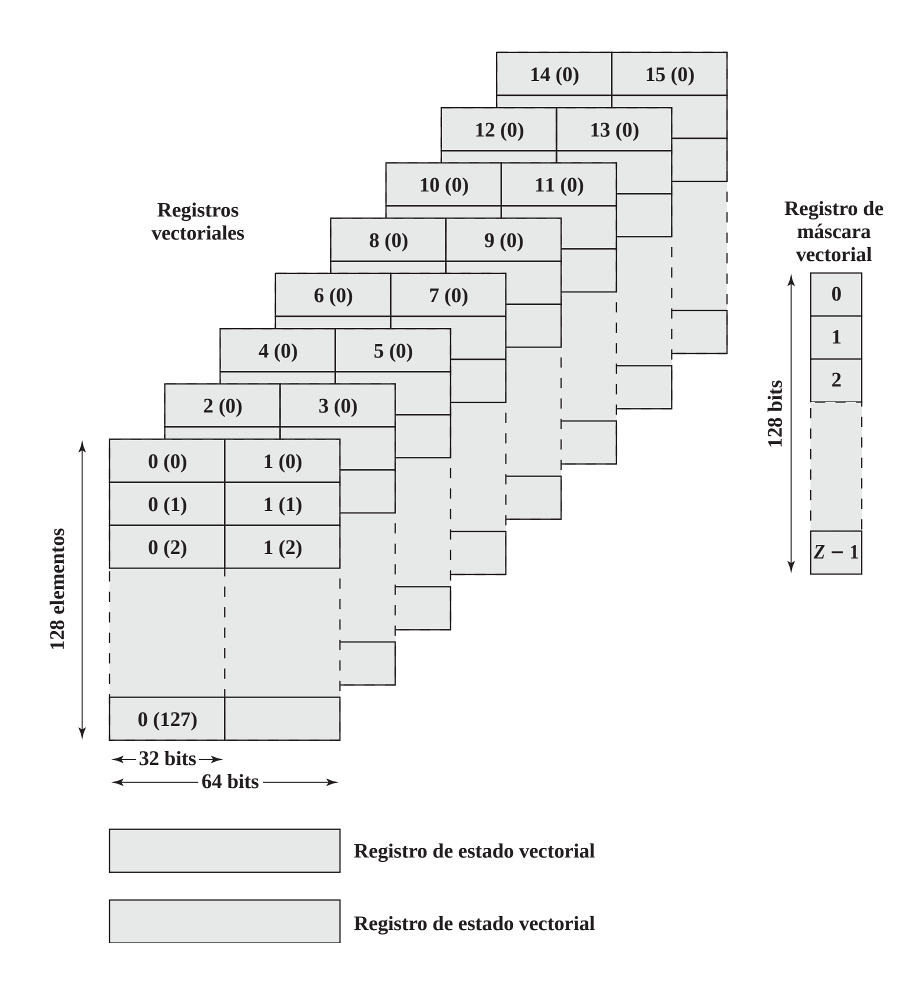


Figura 18.20. Registros de la unidad vectorial del IBM3090.

[5] Las instrucciones compuestas, discutidas más adelante, proporcionan una reducción adicional.

operación vectorial consta esencialmente de la sobrecarga (*overhead*) debida al tiempo de latencia de inicio del cauce (*pipeline starup*) y del llenado de los registros, más un ciclo por elemento vectorial. Por tanto, el uso de un número elevado de elementos en cada registro reduce la importancia relativa del tiempo de inicio en el tiempo total de cómputo. No obstante, esta eficiencia debe contrapesarse con el tiempo extra que se necesita para guardar y restaurar los registros vectoriales al conmutar un proceso, y con las limitaciones de costo y espacio. Estas consideraciones han motivado el uso de 128 elementos por registro en la implementación actual del 3090.

La unidad vectorial necesita tres registros adicionales. El registro de máscara de vector (*vector-mask*) contiene los bits de máscara que pueden utilizarse para seleccionar qué elementos de un registro vectorial se procesan en una operación concreta. El registro de estado vectorial (*vector-status*) contiene los campos de control, tales como la cuenta de vector que determina cuántos elementos de los registros vectoriales se van a procesar. La cuenta de actividad vectorial (*vector-activity*) almacena el tiempo transcurrido ejecutando instrucciones vectoriales.

Instrucciones compuestas. Como se discutió arriba, para mejorar las prestaciones se puede solapar la ejecución de varias instrucciones utilizando encadenamiento. Los diseñadores de la unidad vectorial IBM prefirieron no incluir esta capacidad por varias razones. La arquitectura del S/370 tendría que extenderse para manejar interrupciones complejas (incluyendo sus efectos en la gestión de la memoria virtual), y se necesitarían los correspondientes cambios en el software. Una cuestión más importante fue el costo de incluir las señales de control adicionales y los caminos de acceso a los registros en la unidad vectorial para conseguir el encadenamiento generalizado.

En su lugar, se incluyeron tres operaciones que combinan en una instrucción (un solo código de operación) las secuencias más comunes en los cálculos vectoriales, concretamente la multiplicación seguida por una suma, resta o acumulación. La instrucción MULTIPLY-AND-ADD de memoria-a-registro, por ejemplo, capta un vector de la memoria, lo multiplica por un vector almacenado en un registro y suma el producto a un tercer vector en otro registro. Utilizando las instrucciones compuestas MULTIPLY-AND-ADD y MULTIPLY-AND-SUBSTRACT en el ejemplo de la Figura 18.19, el tiempo total de una iteración se reduce de diez a ocho ciclos.

A diferencia del encadenamiento, las instrucciones compuestas no precisan utilizar registros adicionales para el almacenamiento temporal de los resultados intermedios, y requieren un acceso a registro menos. Por ejemplo, considere la siguiente cadena:

$$A \rightarrow VR1$$
$$VR1 + VR2 \rightarrow VR1$$

En este caso, se requieren dos almacenamientos en el registro vectorial VR1. En la arquitectura del IBM hay una instrucción ADD de memoria-a-registro. Con esta instrucción, solo la suma final se sitúa en VR1. La instrucción compuesta además evita la necesidad de reflejar en la descripción del estado de la máquina la ejecución concurrente de varias instrucciones, con lo que se simplifica el almacenamiento y la recuperación del estado por parte del sistema operativo y la gestión de interrupciones.

El conjunto de instrucciones. La Tabla 18.3 presenta las operaciones lógicas y aritméticas definidas para la arquitectura vectorial. Además, hay instrucciones de carga de memoria-a-registro y de almacenamiento de registro-a-memoria. Obsérvese que muchas instrucciones utilizan un formato de tres operandos. Además, muchas instrucciones poseen un cierto número de variaciones, según la

Tabla 18.3. Unidad vectorial del IBM; instrucciones lógicas y aritméticas.

<table>
<tr><th rowspan="3">Operación</th><th colspan="3">Tipos de datos</th><th colspan="4" rowspan="3">Posiciones de los operandos</th></tr>
<tr><th colspan="2">Punto flotante</th><th rowspan="2">Binario o lógico</th></tr>
<tr><th>Largo</th><th>Corto</th></tr>
<tr><td>Add</td><td>FL</td><td>FS</td><td>BI</td><td>V + V → V</td><td>V + S → V</td><td>Q + V → V</td><td>Q + S → V</td></tr>
<tr><td>Subtrat</td><td>FL</td><td>FS</td><td>BI</td><td>V − V → V</td><td>V − S → V</td><td>Q − V → V</td><td>Q − C → V</td></tr>
<tr><td>Multiply</td><td>FL</td><td>FS</td><td>BI</td><td>V × V → V</td><td>V × V → V</td><td>Q × V → V</td><td>Q × S → V</td></tr>
<tr><td>Divide</td><td>FL</td><td>FS</td><td>—</td><td>V / V → V</td><td>V / S → V</td><td>Q / V → V</td><td>Q / S → V</td></tr>
<tr><td>Compare</td><td>FL</td><td>FS</td><td>BI</td><td>V · V → V</td><td>V · S → V</td><td>Q · V → V</td><td>Q · S → V</td></tr>
<tr><td>Multiply and Add</td><td>FL</td><td>FS</td><td>—</td><td></td><td>V + V · S → V</td><td>V + Q × V → V</td><td>V + Q · S → V</td></tr>
<tr><td>Multiply and Subtract</td><td>FL</td><td>FS</td><td>—</td><td></td><td>V − V × S → V</td><td>V − Q × V → V</td><td>V − Q × S → V</td></tr>
<tr><td>Multiply and Accumulate</td><td>FL</td><td>FS</td><td>—</td><td>P + · V → V</td><td>P + · S → V</td><td></td><td></td></tr>
<tr><td>Complement</td><td>FL</td><td>FS</td><td>BI</td><td>− V → V</td><td></td><td></td><td></td></tr>
<tr><td>Positive Absolute</td><td>FL</td><td>FS</td><td>BI</td><td>|V| → V</td><td></td><td></td><td></td></tr>
<tr><td>Negative Absolute</td><td>FL</td><td>FS</td><td>BI</td><td>− |V| → V</td><td></td><td></td><td></td></tr>
<tr><td>Maximum</td><td>FL</td><td>FS</td><td>—</td><td></td><td></td><td>Q · V → Q</td><td></td></tr>
<tr><td>Maximum Absolute</td><td>FL</td><td>FS</td><td>—</td><td></td><td></td><td>Q · V → Q</td><td></td></tr>
<tr><td>Minimum</td><td>FL</td><td>FS</td><td>—</td><td></td><td></td><td>Q · V → Q</td><td></td></tr>
<tr><td>Shift Left Logical</td><td>—</td><td>—</td><td>LO</td><td>· V → V</td><td></td><td></td><td></td></tr>
<tr><td>Shift Right Logical</td><td>—</td><td>—</td><td>LO</td><td>· V → V</td><td></td><td></td><td></td></tr>
<tr><td>And</td><td>—</td><td>—</td><td>LO</td><td>V & V → V</td><td>V & S → V</td><td>Q & V → V</td><td>Q & S → V</td></tr>
<tr><td>Or</td><td>—</td><td>—</td><td>LO</td><td>V|V → V</td><td>V|S → V</td><td>Q|V → V</td><td>Q|S → V</td></tr>
<tr><td>Exclusive-Or</td><td>—</td><td>—</td><td>LO</td><td>V ≠ V → V</td><td>V ≠ S → V</td><td>Q ≠ V → V</td><td>Q ≠ S → V</td></tr>
</table>

Explicación:

Tipos de datos
FL = punto-flotante largo
FS = punto-flotante corto
Bi = entero binario
LO = lógico

Posiciones de los operandos
V = registro vectorial
S = memoria
Q = escalar (general o registro de punto-flotante)
· = operación especial

situación de los operandos. Un operando fuente puede ser un registro vectorial (V), un registro escalar (Q), o estar en memoria (S, *storage*). El destino es siempre un registro vectorial, excepto en la comparación, cuyo resultado va al registro de máscara vectorial (*vector-mask*). Con todas estas variantes, el número total de códigos de operación (instrucciones distintas) es 171. Este considerable número de instrucciones, sin embargo, no es tan costoso de implementar como podría imaginarse. Las unidades funcionales y los caminos de datos para proporcionar operandos a los cauces vectoriales desde la memoria y los registros vectoriales, son los principales responsables del costo del hardware. La arquitectura puede, con pocas diferencias en el coste, proporcionar un conjunto amplio de variantes en el uso de estos registros y cauces.

La mayoría de las instrucciones de la Tabla 18.3 son autoexplicativas. Las dos instrucciones de acumulación requieren una explicación adicional. La operación de acumulación suma todos los elementos de un vector (ACCUMULATE) o los elementos del producto de dos vectores (MULTIPLY-AND-ACCUMULATE). Estas instrucciones plantean un problema de diseño interesante. Nos gustaría realizar esta operación tan rápidamente como sea posible, aprovechando las ventajas del cauce segmentado de la ALU. La dificultad está en que la suma de dos números que se introducen en el cauce no está disponible hasta varios ciclos más tarde. Como consecuencia, el tercer elemento del vector no se podría añadir a la suma de los dos primeros hasta que estos no hayan pasado a través de todo el cauce. Para superar este problema, los elementos del vector se suman de forma que produzcan cuatro sumas parciales. Concretamente, con los elementos 0,4,8,12...,124 se obtiene la suma parcial 0; con los elementos 1,5,9,13...,125 la suma parcial 1; con los elementos 2,6,10,14...,126 la suma parcial 2; y con los elementos 3,7,11,15...,127 la suma parcial 3. Cada una de estas sumas parciales puede realizarse en el cauce a la máxima velocidad, puesto que el retardo del cauce es de cuatro ciclos. Un registro vectorial distinto se utiliza para almacenar las sumas parciales. Cuando se han procesado todos los elementos del vector original, se suman las cuatro sumas parciales para obtener el resultado final. El tiempo de esta segunda fase no es crítico, puesto que solo están implicados cuatro elementos.

18.8. LECTURAS RECOMENDADAS

[CATA94] revisa los principios de los multiprocesadores y examina con detalle los multiprocesadores basados en el SPARC. Los SMP también se tratan en [STON93] y [HWAN93].

Una revisión de los algoritmos y técnicas de coherencia de caché para multiprocesadores se tiene en [MILE00], donde se hace énfasis en los aspectos relacionados con las prestaciones. Otra revisión de las cuestiones relativas a la coherencia de caché en multiprocesadores es [LILJ93]. En [TOMA93] se pueden encontrar reimpresiones de muchos de los artículos clave sobre el tema.

Una revisión excelente de los conceptos relacionados con los procesadores multihebra es [UNGE02]. [UNGE03] es una revisión algo más extensa de los procesadores que utilizan multihebra explícita, tanto actuales como de los propuestos.

[PFIS98] es el libro a leer para cualquiera que esté interesado en los clusters; el libro cubre los aspectos de diseño de hardware y el software y compara los clusters con los SMP y los NUMA; el libro también contiene una sólida descripción técnica de las cuestiones de diseño de los SMP y los NUMA. Un minucioso tratamiento de los clusters se puede encontrar en [BUYY99a] y [BUYY99b]. En [WEY01] se tiene una revisión menos técnica de los clusters, con buenos comentarios acerca de diferentes productos comerciales.

En [STON93] y [HWAN93] se pueden encontrar buenas discusiones de la computación vectorial.

BUYY99a Buyya, R.: *High Performance Cluster Computing: Architectures and Systems.* Upper Saddle River, NJ. Prentice Hall, 1999.

BUYY99b Buyya, R.: *High Performance Cluster Computing: Programming and Applications.* Upper Saddle River, NJ. Prentice Hall, 1999.

CATA94 Catanzaro, B.: *Multiprocessor System Architectures.* Mountain View, CA. Sunsoft Press, 1994.

HWAN93 Hwang, K.: *Advanced Computer Architectura.* New York. McGraw-Hill, 1993.

LILJ93 Lilja, D.: «Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons». *ACM Computing Surveys,* septiembre 1993.

MILE00 Milenkovic, A.: «Achieving High Performance in Bus-Based Shared-Memory Multiprocessors». *IEEE Concurrency,* julio-septiembre, 2000.

PFIS98 Pfister, G.: *In Search of Clusters.* Upper Saddle River, NJ. Prentice Hall, 1998.

STON93 Stone, H.: *High-Performance Computer Architecture.* Reading, MA. Addison-Wesley, 1993.

TOMA93 Tomasevic, M. y Milutinovic, V.: *The Cache Coherence Problem in Shared-Memory Multiprocessors: Hardware Solutions.* Los Alamos, CA. IEEE Computer Society Press, 1993.

UNGE02 Ungerer, T.; Rubic, B. y Silc, J.: «Multithreaded Processors». *The Computer Journal,* No. 3, 2002.

UNGE03 Ungerer, T.; Rubic, B. y Silc, J.: «A survey of Processors with Explicit Multithreading». *ACM Computing Surveys,* marzo, 2003.

WEYG01 Weygant, P.: *Clusters for Higt Availability.* Upper Saddle River, NJ. Prentice Hall, 2001.



SITIOS WEB RECOMENDADOS

- **IEEE Computer Society Task Force on Cluster Computing:** un foro internacional para promover la investigación y educación en los aspectos relacionados con la computación en *clusters.*

18.9. PALABRAS CLAVE, CUESTIONES Y PROBLEMAS

PALABRAS CLAVE

acceso no uniforme a memoria (NUMA)	espera pasiva	protocolo MESI
acceso uniforme a memoria (UMA)	monoprocesador	recuperación después de un fallo *(failback)*
coherencia de caché	multiprocesador	transferencia por fallo *(failover)*
cluster	multiprocesador simétrico (SMP)	unidad vectorial
espera activa	protocolo de directorio	
	protocolo de sondeo	

CUESTIONES

18.1. Enumere y defina brevemente tres tipos de organización del computador.

18.2. ¿Cuáles son las características principales de un SMP?

18.3. ¿Cuáles son algunas de las ventajas potenciales de un SMP comparado con un computador monoprocesador?

18.4. ¿Cuáles son los aspectos clave en el diseño del sistema operativo de un SMP?

18.5. ¿Cuál es la diferencia entre un esquema de coherencia de caché hardware y uno software?

18.6. ¿Qué significan cada uno de los cuatro estados del protocolo MESI?

18.7. Indique algunos de los beneficios más importantes de los *clusters.*

18.8. ¿Cuál es la diferencia entre transferencia por fallo (*failover*) y recuperación después de un fallo (*failback*)?

18.9. ¿Qué diferencias existen entre UMA, NUMA, y CC-NUMA?

PROBLEMAS

18.1. Sea α el porcentaje de código de programa que puede ejecutarse simultáneamente por los n procesadores de un computador. Asuma que el resto del código debe ejecutarse secuencialmente por un solo procesador. Cada procesador tiene una velocidad de ejecución de x MIPS.

(a) Proporcione una expresión para los MIPS efectivos en función de n, α y x, cuando se utiliza este sistema para ejecutar exclusivamente este programa.

(b) Si $n = 16$ y $x = 4$ MIPS, determine el valor de α que hace que las prestaciones del sistema sean iguales a 40 MIPS.

18.2. Un multiprocesador con ocho procesadores tiene conectadas veinte unidades de cinta. Hay un gran número de trabajos enviados al sistema, y cada uno de ellos necesita un máximo de cuatro unidades de cinta para completar su ejecución. Asuma que cada trabajo comienza a ejecutarse utilizando tres unidades de cinta durante un periodo largo antes de que necesite la cuarta unidad de cinta, durante un corto periodo de tiempo antes de finalizar su ejecución. Asuma también una fuente continua que suministra trabajos.

(a) Suponga que el planificador del sistema operativo no iniciará ningún trabajo a no ser que existan cuatro unidades de cinta disponibles. Cuando un trabajo comienza, inmediatamente se le asignan cuatro unidades de cinta y no se liberan hasta que el trabajo finalice. ¿Cuál es el número máximo de trabajos que pueden estar ejecutándose al mismo tiempo? ¿Cuál es el número máximo y mínimo de unidades de cinta que pueden estar inactivas como resultado de esta política?

(b) Sugiera una política alternativa que mejore la utilización de las unidades de cinta al mismo tiempo que se evita el bloqueo *(deadlock)* del sistema. ¿Cuál es el número máximo de trabajos que pueden estar ejecutándose al mismo tiempo? ¿Cuáles son los límites en el número de unidades de cinta inactivas?

18.3. ¿Puede existir algún problema con la aproximación de caché *write-once* en los multiprocesadores basados en bus? Si es así, sugiera una solución.

18.4. Considere que dos procesadores de un SMP necesitan acceder a la misma línea de datos de memoria principal. Ambos procesadores tienen caché y utilizan el protocolo MESI. Inicialmente, ambas cachés tienen una copia no válida de la línea. La Figura 18.21 muestra el resultado de la lectura de la línea x por parte del procesador P1. Si este es el inicio de una secuencia de accesos, dibuje las figuras correspondientes a la siguiente secuencia:

1. P2 lee x.
2. P1 escribe en x (por claridad, marque con x' la línea en la caché de P1).
3. P1 escribe en x (marque con x'' la línea de caché en P1).
4. P2 lee x.

18.5. La Figura 18.22 muestra dos diagramas de estados posibles como protocolos de coherencia de caché. Deduzca y explique cada protocolo y compárelo con el protocolo MESI.

18.6. Considere un SMP con cachés L1 y L2 que utilizan el protocolo MESI. Como se explica en la Sección 18.3, cada línea de la caché L2 puede estar en uno de los cuatro estados del protocolo. ¿Se necesitan todos los estados en cada línea de la caché L1? Si es así, ¿por qué? Si no, explique qué estado o estados se pueden eliminar.

18.7. Una primera versión del S/390 de IBM, el S/390 G4, utilizaba tres niveles de caché. Como en el z990, solo el primer nivel se encontraba en el chip de procesador —denominado unidad procesadora (PU)—. La caché L2 también era similar a la del z990. La caché L3 se encontraba en un chip distinto

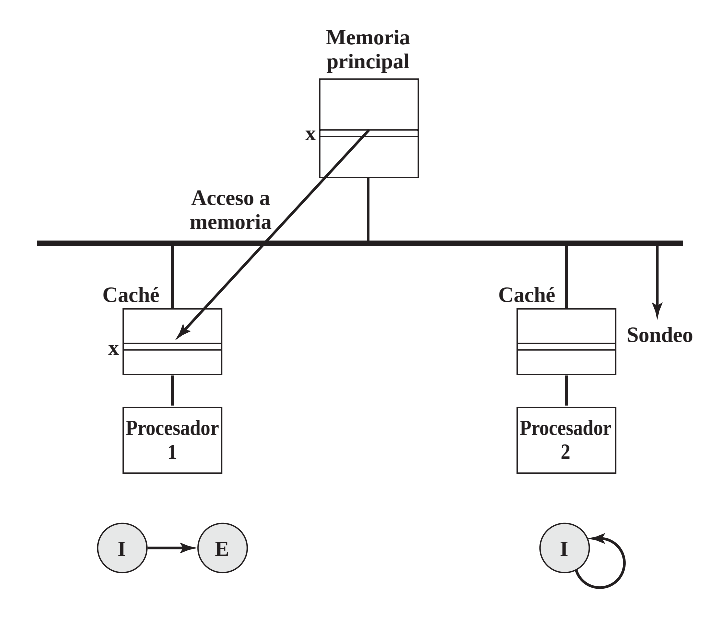


Figura 18.21. Ejemplo MESI: el procesador 1 lee la línea x.

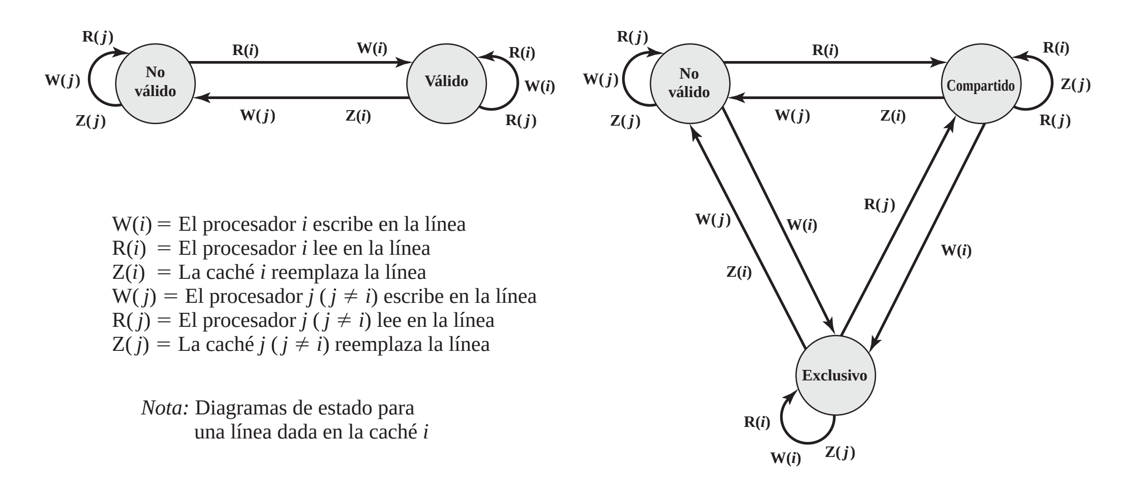


Figura 18.22. Protocolos de coherencia para dos cachés.

que actuaba como controlador de memoria, y que estaba entre las cachés L2 y los módulos de memoria. La Tabla 18.4 muestra las prestaciones de la disposición de tres niveles de caché del IBM S/390. El propósito de este problema es determinar si la inclusión de un tercer nivel de caché resulta beneficiosa. Determine la penalización de acceso (promedio de ciclos de PU) para un sistema con una única caché L1 y normalice este valor a 1.0. Después determine la penalización de acceso normalizada cuando se utilizan las cachés L1 y L2, y la penalización de acceso al utilizar las tres cachés. Considere la magnitud de la mejora en cada caso e indique su opinión sobre el interés de utilizar la caché L3.

18.8. (a) Un computador monoprocesador tiene cachés separadas para instrucciones y datos con porcentajes de aciertos H_s y H_i, respectivamente. El tiempo de acceso del procesador a la caché es de *c* ciclos de reloj, y el tiempo de transferencia de un bloque entre memoria y caché es de *b* ciclos de

Tabla 18.4. Porcentajes de acierto típicos en la configuración de memoria del SMP S/390 [MAK97].

Subsistema de memoria	Penalización de acceso (ciclos de PU)	Tamaño de la caché	Acierto (%)
caché L1	1	32 KB	89
caché L2	5	256 KB	5
caché L3	14	2 MB	3
Memoria	32	8 GB	3

reloj. Sea f_i la fracción de accesos a memoria correspondientes a instrucciones, y f_d la fracción de accesos a líneas modificadas respecto a las líneas reemplazadas en la caché de datos. Considere una política de postescritura y determine el tiempo efectivo de acceso a memoria en términos de los parámetros definidos.

(b) Ahora, considere un SMP basado en un bus en el que cada procesador tiene las características descritas en el apartado (a). Cada procesador debe gestionar la invalidación de caché además de las lecturas y escrituras en memoria. Esto afecta al tiempo efectivo de acceso a memoria. Sea f_{inv} la fracción de referencias a datos que ocasionan señales de invalidación a otras cachés de datos. El procesador que envía la señal necesita t ciclos de reloj para completar la operación de invalidación. Ningún otro procesador interviene en la operación de invalidación. Determine el tiempo efectivo de acceso a memoria.

18.9. ¿Qué alternativa de organización corresponde a cada una de las ilustraciones de la Figura 18.23?

18.10. En la Figura 18.8, algunos de los diagramas presentan filas que están parcialmente sombreadas. En otros casos hay filas que están completamente tachadas. Estas dos situaciones representan dos tipos distintos de pérdida de eficiencia. Explíquelos.

18.11. El cauce de la Figura 12.13b se ha redibujado en la Figura 18.24a, ignorando las etapas de captación y decodificación, para representar la ejecución de la hebra A. La Figura 18.24b muestra la ejecución de una hebra B, distinta. En ambos casos se utiliza un procesador segmentado.

(a) Muestre un diagrama de emisión de instrucciones similar al de la Figura 18.8a para cada una de las dos hebras.

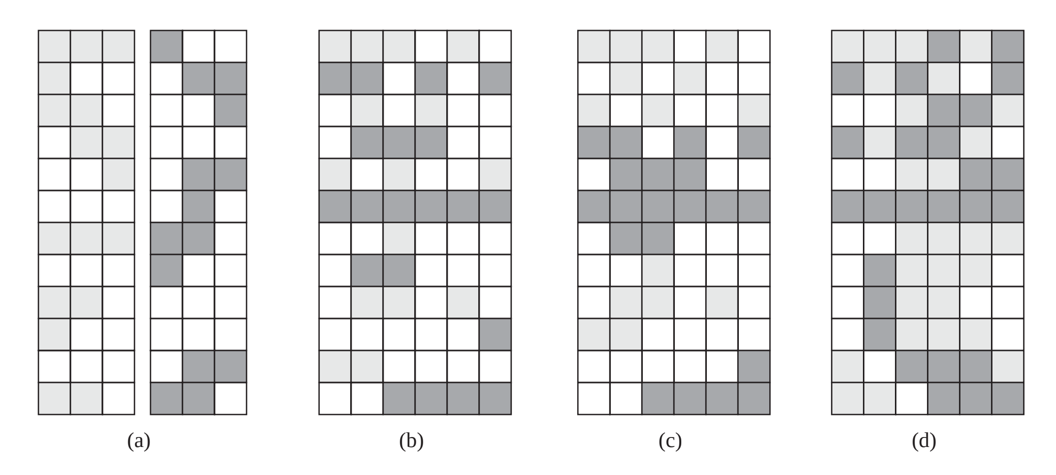
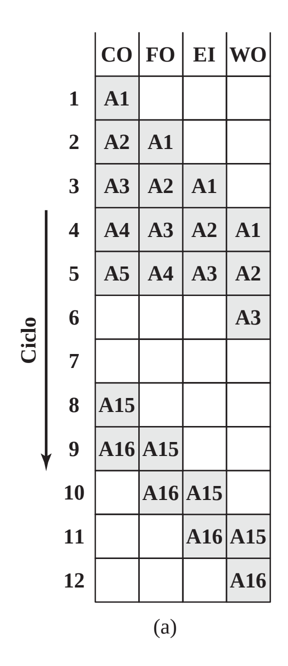


Figura 18.23. Diagrama para el problema 18.9.



	CO	FO	EI	WO
1	B1			
2	B2	B1		
3	B3	B2	B1	
4	B4	B3	B2	B1
5			B3	B2
6				B3
7	B5	B4		
8	B6	B5	B4	
9	B7	B6	B5	B4
10		B7	B6	B5
11			B7	B6
12				B7

(b)

Figura 18.24. Dos hebras de ejecución.

(b) Considere que las dos hebras se ejecutan en paralelo en un multiprocesador monochip en el que cada uno de los procesadores del chip es un procesador segmentado. Muestre un diagrama de emisión de instrucciones similar al de la Figura 18.8k. Además, dibuje un diagrama de ejecución en el cauce del estilo de la Figura 18.24.

(c) Considere una arquitectura superescalar de dos vías de emisión. Repita el apartado (b) para una implementación superescalar multihebra entrelazada, suponiendo que no hay dependencias de datos. *Nota:* no existe una respuesta única; necesita realizar suposiciones sobre los retardos y las prioridades.

(d) Repita el apartado c para una implementación superescalar multihebra con bloqueo.

(e) Repítala para una arquitectura SMT de cuatro vías.

18.12. El siguiente segmento de código debe ejecutarse 64 veces para evaluar la expresión aritmética: $D(I) = A(I) + B(I) \times C(I)$ para $0 \leq I \leq 63$.

Load R1, B(I) / R1 $\leftarrow$ Memoria $(\alpha + I)$/
Load R2, C(I) / R2 $\leftarrow$ Memoria $(\beta + I)$/
Mult R1, R2 / R1 $\leftarrow$ (R1) $\times$ (R2)/
Load R3, A(I) / R3 $\leftarrow$ Memoria $(\gamma + I)$/
Add R3, R1 / R3 $\leftarrow$ (R3) + (R1)/
Load D1, R3 / Memoria $(\theta + I) \leftarrow$ (R3)/

donde R1, R2, y R3 son registros del procesador, y α, β, γ, y θ son las direcciones de memoria principal de comienzo de los vectores B(I), C(I), A(I) y D(I), respectivamente. Asuma cuatro ciclos de reloj para una operación de carga (Load) o almacenamiento (Store), dos ciclos para la suma (Add) y ocho para la multiplicación (Mult), tanto en el procesador de un computador monocomputador como en el de una máquina SIMD.

(a) Calcule el número total de ciclos de reloj que se necesitan para ejecutar este segmento de código 64 veces en un computador monoprocesador, SISD, ignorando todos los otros retardos.

(b) Considere que se utiliza un SIMD con 64 elementos de proceso para ejecutar operaciones vectoriales utilizando seis instrucciones vectoriales, sincronizadas por el mismo reloj, y que actuan sobre vectores de datos de 64 componentes. Calcule el tiempo de ejecución total en la máquina SIMD ignorando la instrucción de difusión (*broadcast*) y los demás retardos.

(c) Cual es la ganancia de velocidad (*speedup*) del computador SIMD con respecto al SISD.

18.13. Obtenga la versión vectorizada del siguiente programa:

```
     DO 20 I = 1,N
     B(I, 1) = 0
     DO 10 J = 1, M
     A(I) = A(I) + B(I, J) × C(I, J)
10   CONTINUE
     D(I) = (E(I) + A(I))
20   CONTINUE
```

18.14. Un programa se ejecuta en un *cluster* con nueve computadores. Se trata de un programa de prueba *(benchmark)* que consume un tiempo *T* en este *cluster.* De este tiempo *T,* el programa estuvo ejecutándose en los nueve computadores un 25 por ciento. El resto del tiempo se ejecutó en un sólo computador.

(a) Calcule la ganancia de velocidad efectiva en las condiciones mencionadas con respecto a la ejecución de todo el programa en un solo computador. Calcule también α, porcentaje de código que ha sido paralelizado (programado o compilado para utilizar el *cluster*) en el programa indicado.

(b) Suponga que se pueden utilizar 18 procesadores en lugar de nueve en la parte de código paralelizada. Calcule la ganancia de velocidad efectiva que se consigue.

18.15. El siguiente programa FORTRAN se ejecuta en un computador, y una versión paralela del mismo en un cluster con 32 computadores.

```
L1:      DO 10 I = 1, 1024
L2:           SUM(I) = 0
L3:           DO 20 J = 1, I
L4:  20            SUM(I) = SUM(I) + I
L5:  10  CONTINUE
```

Suponga que las líneas 2 y 4 tardan en ejecutarse dos ciclos máquina cada una incluyendo todas las acciones del procesador y los accesos a memoria. Ignore la penalización originada por las sentencias de control del bucle (líneas 1, 3, 5) y todas las otras fuentes de penalización y conflictos entre los recursos.

(a) ¿Cuál es el tiempo total de ejecución (en ciclos máquina) del programa en un único computador?

(b) Divida las I iteraciones del bucle entre los 32 computadores tal y como se indica a continuación: el computador 1 ejecuta las primeras 32 iteraciones (I = 1 a 32), el procesador 2 ejecuta las siguientes 32, y así sucesivamente. ¿Cuál es el tiempo de ejecución y la ganancia de velocidad con relación al apartado (a)? (tenga en cuenta que la carga de trabajo definida por el bucle J no está igualmente distribuida entre los computadores).

(c) Explique cómo modificaría la paralelización anterior para conseguir una ejecución con la carga de trabajo distribuida de forma equilibrada entre los 32 computadores. Se entiende que en una carga de trabajo equilibrada se asigna el mismo número de sumas (considerando los dos bucles) a todos los computadores.

(d) ¿Cuál es el tiempo de ejecución mínimo que resulta de la ejecución paralela en los 32 computadores? ¿Cuál es la ganancia de velocidad que se obtiene con respecto a la ejecución en un solo computador?

18.16. Considere las dos versiones siguientes de un programa que suma dos vectores:

L1: **DO** 10 I = 1, N	**DOALL** K = 1, M
L2: A(I) = B(I) + C(I)	**DO** 10 I = L(K − 1) + 1, KL
L3: 10 **CONTINUE**	A(I) = B(I) + C(I)
L4: SUM = 0	10 **CONTINUE**
L5: **DO** 20 J = 1, N	SUM(K) = 0
L6: SUM = SUM + A(J)	**DO** 20 J = 1, L
L7: 20 **CONTINUE**	SUM(K) = SUM(K) + A(L(K − 1) + J)
	20 **CONTINUE**
	ENDALL

(a) El programa de la derecha se ejecuta en un computador monoprocesador. Suponga que las líneas de código L2, L4 y L6 tardan un ciclo de reloj del procesador en ejecutarse. Por simplicidad, ignore el tiempo que necesitan las otras líneas del código. Inicialmente todos los vectores se encuentran en memoria principal y el pequeño fragmento de programa en la caché de instrucciones. ¿Cuántos ciclos hacen falta para ejecutar este programa?

(b) El programa de la derecha se ha escrito para ser ejecutado en un multiprocesador de M procesadores. Las operaciones de los bucles se reparten en M secciones con $L = N/M$ elementos por sección. La sentencia DOALL declara que las M secciones se ejecutan en paralelo. Como resultado de este programa se obtienen M sumas parciales. Considere que se necesitan k ciclos de reloj para cada operación de comunicación entre los procesadores a través de la memoria compartida y que, por tanto, la adición de cada suma parcial necesita k ciclos. Un árbol de sumadores binarios de l niveles puede acumular todas la sumar parciales, siendo $l = \log_2 M$. ¿Cuantos ciclos se necesitan para obtener la suma final?

(c) Suponga que hay $N = 2^{20}$ elementos y $M = 256$. ¿Qué ganancia de velocidad se obtiene utilizando el multiprocesador? Considere que $k = 200$. ¿Qué porcentaje es este de la ganancia de velocidad teórica de un factor de 256?

APÉNDICE A

Sistemas de numeración

A.1. SISTEMA DECIMAL

A diario utilizamos un sistema basado en los dígitos decimales (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) para representar los números; es el denominado sistema decimal. Considere lo que significa el número 83. Significa ocho veces diez, más tres:

$$83 = (8 \times 10) + 3$$

El número 4728 significa cuatro veces mil, siete veces cien, dos veces diez, más ocho:

$$4728 = (4 \times 1000) + (7 \times 100) + (2 \times 10) + 8$$

El sistema decimal se dice que usa la **base** 10. Esto significa que cada dígito del número se multiplica por diez elevado a la potencia correspondiente a la posición de dicho dígito. Así:

$$83 = (8 \times 10^1) + (3 \times 10^0)$$

$$4728 = (4 \times 10^3) + (7 \times 10^2) + (2 \times 10^1) + (8 \times 10^0)$$

Los valores fraccionarios se representan de la misma manera, pero utilizando potencias negativas de diez. Así, el numero 0,256 significa dos décimas más cinco centésimas más seis milésimas:

$$0{,}256 = (2 \times 10^{-1}) + (5 \times 10^{-2}) + (6 \times 10^{-3})$$

Un número con parte entera y parte fraccionaria tiene dígitos ponderados por potencias positivas y negativas de 10:

$$472{,}256 = (4 \times 10^2) + (7 \times 10^1) + (2 \times 10^0) + (2 \times 10^{-1}) + (5 \times 10^{-2}) + (6 \times 10^{-3})$$

En general, para la representación decimal de $X = \{\dots d_2 d_1 d_0 \,.\, d_{-1} d_{-2} d_{-3} \dots\}$, el valor de X es:

$$X = \sum_i (d_i \times 10^i)$$

A.2. SISTEMA BINARIO

En el sistema decimal se emplean diez dígitos diferentes para representar números en base diez. En el sistema binario se tienen solo los dígitos 1 y 0. Por tanto, los números en el sistema binario se representan en base 2.

Para evitar confusiones pondremos a veces un subíndice en los números que indica su base. Por ejemplo, 83_{10} y 4728_{10} son números representados en notación decimal, o simplemente números decimales. El 1 y el 0 en notación binaria tienen el mismo significado que en notación decimal:

$$0_2 = 0_{10}$$

$$1_2 = 1_{10}$$

Para representar números mayores, como ocurre en la notación decimal, cada dígito de un número binario tiene un valor que depende de su posición:

$$10_2 = (1 \times 2^1) + (0 \times 2^0) = 2_{10}$$
$$11_2 = (1 \times 2^1) + (1 \times 2^0) = 3_{10}$$
$$100_2 = (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 4_{10}$$

y así sucesivamente. De forma similar, los valores fraccionarios se representan con potencias negativas de la base:

$$1001{,}101 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9{,}625_{10}$$

En general, para la representación binaria de $Y = \{\ldots\, b_2 b_1 b_0 \,.\, b_{-1} b_{-2} b_{-3} \ldots\}$, el valor de Y es:

$$Y = \sum_i (b_i \times 2^i)$$

A.3. CONVERSIÓN ENTRE BINARIO Y DECIMAL

Es fácil convertir un número de notación binaria a decimal. En la subsección anterior hemos mostrado diversos ejemplos. Todo lo que se necesita es multiplicar cada dígito binario por la potencia de 2 apropiada y sumar los resultados.

Para convertir de decimal a binario, las partes entera y fraccionaria se tratan por separado.

ENTEROS

Para la parte entera, recuerde que en notación binaria un entero representado mediante:

$$b_{m-1}\, b_{m-2} \ldots b_2 b_1 b_0 \qquad b_i = 0 \text{ ó } 1$$

tiene el valor:

$$(b_{m-1} \times 2^{m-1}) + (b_{m-2} \times 2^{m-2}) + \ldots + (b_1 \times 2^1) + b_0$$

Suponga que se quiere convertir un entero decimal N a forma binaria. Si dividimos N entre dos en el sistema decimal, y obtenemos un cociente N_1 y un resto R_0, podemos escribir:

$$N = 2 \times N_1 + R_0 \qquad R_0 = 0 \text{ ó } 1$$

A continuación dividimos el cociente N_1 entre dos. Suponga que el nuevo cociente es N_2 y el nuevo resto es R_1. Entonces:

$$N_1 = 2 \times N_2 + R_1 \qquad R_1 = 0 \text{ ó } 1$$

de manera que:

$$N = 2 \times (2\, N_2 + R_1) + R_0 = (N_2 \times 2^2) + (R_1 \times 2^1) + R_0$$

Si sustituimos

$$N_2 = 2\, N_3 + R_2$$

tenemos:

$$N = (N_3 \times 2^3) + (R_2 \times 2^2) + (R_1 \times 2^1) + R_0$$

Continuando este proceso, ya que $N > N_1 > N_2 \ldots$, llegará a producirse un cociente $N_{m-1} = 1$ (excepto para los enteros decimales 0 y 1, cuyos equivalentes binarios son 0 y 1 respectivamente), y un resto R_{m-2}, que es 0 o 1. Entonces

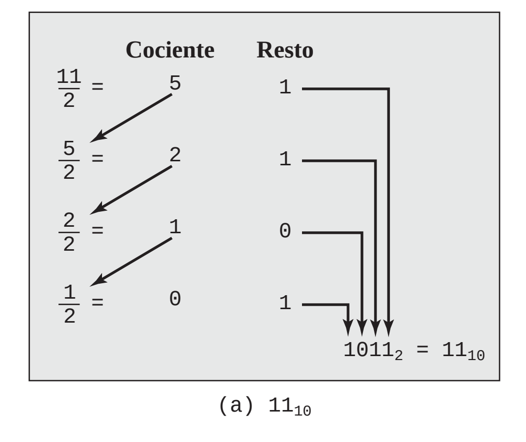
$$N = (1 \times 2^{m-1}) + (R_{m\text{-}2} \times 2^{m-2}) + \ldots + (R_2 \times 2^2) + (R_1 \times 2^1) + R_0$$

que es la forma binaria de N. Es decir, convertimos de base 10 a base 2 mediante divisiones repetidas por dos. Los restos y el cociente final, 1, nos dan los dígitos binarios de N en orden de importancia creciente. La Figura A.1 muestra un par de ejemplos.

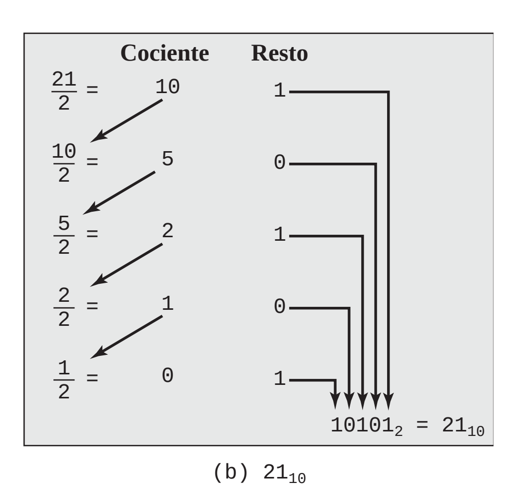
FRACCIONARIOS

Para la parte fraccionaria recuerde que, en notación binaria, un número con un valor entre 0 y 1 se representa mediante:

$$0.\, b_{-1}b_{-2}b_{-3} \ldots \qquad b_i = 0 \text{ ó } 1$$



(a) 11_{10}



(b) 21_{10}

Figura A.1. Ejemplos de conversión de números enteros de la notación decimal a la binaria.

y tiene el valor:

$$(b_{-1} \times 2^{-1}) + (b_{-2} \times 2^{-2}) + (b_{-3} \times 2^{-3}) ...$$

Esto puede rescribirse como:

$$2^{-1} \times (b_{-1} + 2^{-1} \times (b_{-2} + 2^{-1} \times (b_{-3} + ...$$

Esta expresión sugiere un procedimiento para la conversión. Suponga que queremos convertir el número F $(0 < F < 1)$ de notación decimal a binaria. Sabemos que F puede expresarse en la forma:

$$F = 2^{-1} \times (b_{-1} + 2^{-1} \times (b_{-2} + 2^{-1} \times (b_{-3} + ...$$

Si multiplicamos F por 2, obtenemos:

$$2\ (F = b_{-1} + 2^{-1} \times (b_{-2} + 2^{-1} \times (b_{-3} + ...$$

De esta ecuación vemos que la parte entera de $(2 \times F)$, que debe ser 0 o 1 ya que $0 < F < 1$, es simplemente b_{-1}. Por lo tanto, podemos decir $(2 \times F) = b_{-1} + F_1$, en donde $0 < F_1 < 1$ tiene la expresión:

$$F_1 = 2^{-1} \times (b_{-2} + 2^{-1} \times (b_{-3} + 2^{-1} \times (b_{-4} + ...$$

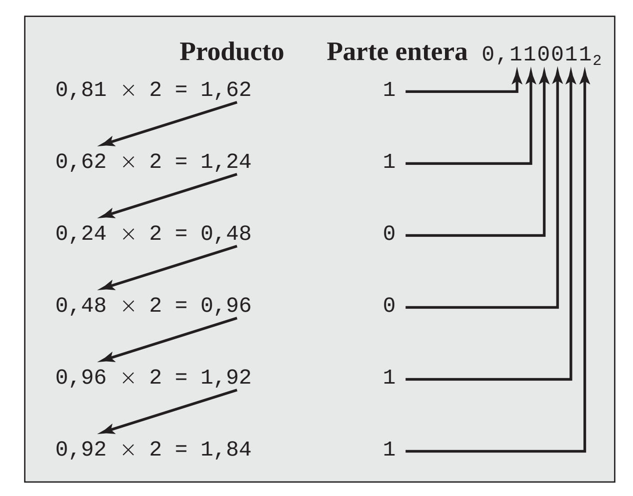
Para encontrar b_{-2} se repite el proceso. En consecuencia, el algoritmo de conversión implica repetidas multiplicaciones por dos. En cada paso se multiplica por dos la parte fraccionaria del número del paso anterior. En el producto resultante, el dígito a la izquierda de la coma decimal será 0 o 1, y pasa a formar parte de la representación binaria, empezando con el dígito más significativo. La parte fraccionaria del producto resultante se usa como multiplicando en el siguiente paso. La Figura A.2 muestra dos ejemplos.

Este proceso no es necesariamente exacto; es decir, una parte fraccionaria decimal con un número finito de dígitos puede requerir una parte fraccionaria binaria con infinitos dígitos. En tales casos, el algoritmo de conversión es normalmente interrumpido después de un número de pasos preestablecido, que depende de la precisión deseada.

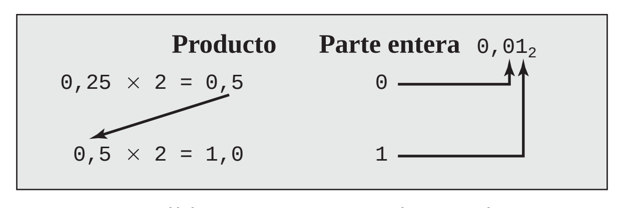
A.4. NOTACIÓN HEXADECIMAL

Dada la naturaleza binaria de los componentes de los computadores digitales, todos los tipos de datos son representados en los computadores mediante diversos códigos binarios. Sin embargo, aunque el sistema binario sea tan adecuado para los computadores, para los humanos resulta altamente engorroso. Como consecuencia, la mayoría de los profesionales de la informática que tienen que trabajar a menudo con los datos en bruto del computador prefieren una notación más compacta.

¿Qué notación utilizar? Una posibilidad es la decimal. Esta es ciertamente más compacta que la notación binaria, pero es engorroso por lo tedioso de convertir entre base 2 y base 10.



(a) $0.81_{10} = 0.11011_2$ (aproximado)



(b) $0.25_{10} = 0.01_2$ (exacto)

Figura A.2. Ejemplos de conversión de números fraccionarios de la notación decimal a la binaria.

En su lugar se ha optado por una notación conocida como hexadecimal. Los dígitos binarios son agrupados en conjuntos de cuatro. A cada combinación posible de cuatro dígitos binarios se asocia un símbolo de la siguiente manera:

0000 = 0	0100 = 4	1000 = 8	1100 = C
0001 = 1	0202 = 5	1001 = 9	1101 = D
0010 = 2	0110 = 6	1010 = A	1110 = E
0011 = 3	0111 = 7	1011 = B	1111 = F

La notación se denomina **hexadecimal** por utilizar 16 símbolos, y a cada uno de ellos se le llama **dígito hexadecimal**.

Una secuencia de dígitos hexadecimales puede considerarse como un entero representado en base 16. Así:

$$2C_{16} = (2_{16} \times 16^1) + (C_{16} \times 16^0) = (2_{10} \times 16^1) + (12_{10} \times 16^0) = 44$$

Pero la notación hexadecimal no se utiliza solo para representar enteros. Se emplea como notación concisa para representar cualquier secuencia de dígitos binarios, ya represente texto, números o cualquier otro tipo de datos. Las razones para utilizar la notación hexadecimal son:

1. Es más compacta que la notación binaria.
2. En la mayoría de los computadores, los datos binarios ocupan múltiplos de cuatro bits, y por tanto múltiplos de un dígito decimal.
3. Es extremadamente fácil convertir entre binario y hexadecimal.

Como ejemplo del último punto, considere la cadena binaria 110111100001. Su equivalente es:

$$\begin{array}{cccl} 1101 & 1110 & 0001 & = \mathrm{DE1}_{16} \\ \mathrm{D} & \mathrm{E} & 1 & \end{array}$$

Este proceso se realiza de forma tan natural que un programador experimentado puede convertir mentalmente las representaciones visuales de los datos binarios a su equivalente hexadecimal sin necesidad de escribirlos.

A.5. PROBLEMAS

A.1. Convierta los siguientes números binarios a sus equivalentes decimales:
a. 001100 b. 000011 c. 011100 d. 111100 e. 101010

A.2. Convierta los siguientes números binarios a sus equivalentes decimales:
a. 11100,011 b. 110011,10011 c. 1010101010,1

A.3. Convierta los siguientes números decimales a sus equivalentes binarios:
a. 64 b. 100 c. 111d. 145 e. 255

A.4. Convierta los siguientes números decimales a sus equivalentes binarios:
a. 34,75 b. 25,25 c. 27,1875

A.5. Exprese los siguientes números octales en notación hexadecimal:
a. 12 b. 5655 c. 2550276 d. 76545336 e. 3726755

A.6. Convierta los siguientes números hexadecimales a sus equivalentes decimales:
a. C b. 9F c. D52 d. 67E e. ABCD

A.7. Convierta los siguientes números hexadecimales a sus equivalentes decimales:
a. F,4 b. D3,E c. 1111,1 d. 888,8 e. EBA,C

A.8. Convierta los siguientes números decimales a sus equivalentes hexadecimales:
a. 16 b. 80 c. 2560 d. 3000 e. 62.500

A.9. Convierta los siguientes números decimales a sus equivalentes hexadecimales:
a. 204,125 b. 255,875 c. 631,25 d. 10000,00390625

A.10. Convierta los siguientes números hexadecimales a sus equivalentes binarios:
a. E b. 1C c. A64 d. 1F,C e. 239,4

A.11. Convierta los siguientes números binarios a sus equivalentes hexadecimales:
a. 1001,1111 b. 11010,011001 c. 10100111,11011

A.12. Demuestre que todo número real con una representación binaria limitada (con un número finito de dígitos a la derecha de la coma binaria) tiene también una representación decimal limitada (número finito de dígitos a la derecha de la coma decimal).

APÉNDICE B

Lógica digital

El funcionamiento de los computadores digitales se basa en la memorización y procesamiento de datos binarios. A lo largo de este libro, hemos supuesto la existencia de elementos de memoria que pueden estar en uno de dos estados estables y de circuitos que pueden operar con datos binarios bajo la acción de señales de control para implementar distintas funciones. En este apéndice, sugerimos cómo se pueden implementar estos elementos de memoria y circuitos, en lógica digital, concretamente con circuitos combinacionales y secuenciales. El apéndice comienza con un breve repaso del álgebra de Boole, que es el fundamento matemático de la lógica digital. Luego presentaremos el concepto de puerta. Finalmente, se describen los circuitos combinacionales y secuenciales, que se construyen con puertas.

B.1. ÁLGEBRA DE BOOLE

La circuitería digital en computadores digitales y otros sistemas digitales, se diseña y se analiza con el uso de una disciplina matemática denominada *álgebra de Boole.* El nombre es en honor al matemático inglés George Boole, que propuso los principios básicos de este álgebra en 1854 en su tratado, *An investigation of the laws of thought on which to found the mathematical theories of logic and probabilities.* En 1938, Claude Shannon, un investigador asistente en el departamento de Ingeniería Eléctrica del M.I.T., sugirió que el álgebra de Boole podría usarse para resolver problemas de diseño de circuitos de conmutación [SHAN38]. Las técnicas de Shannon se usaron, consecuentemente, en el análisis y diseño de circuitos electrónicos digitales. El álgebra de Boole resulta ser una herramienta útil en dos áreas:

- **Análisis:** es una forma concisa de describir el funcionamiento de los circuitos digitales.
- **Diseño:** dada una función deseada, se puede aplicar el álgebra de Boole para desarrollar una implementación de complejidad simplificada de esta función.

Como con cualquier álgebra, el álgebra de Boole usa variables y operaciones. En este caso, las variables y las operaciones son lógicas. Por tanto, una variable puede tomar el valor 1 (VERDADERO) o 0 (FALSO). Las operaciones lógicas básicas son AND, OR y NOT, que se representan simbólicamente con los signos punto, más y rayado superior:

$$\begin{aligned} \text{A AND B} &= \text{A} \cdot \text{A} \\ \text{A OR} &= \text{A} + \text{B} \\ \text{NOT A} &= \overline{\text{A}} \end{aligned}$$

La operación AND es verdadera (valor binario 1) si y solo si los dos operandos son verdaderos. El resultado de la operación OR es verdad si y solo si uno o ambos operandos son verdad. La operación unitaria NOT invierte el valor del operando. Por ejemplo, consideremos la ecuación

$$D = A + (\overline{B} \cdot C)$$

D es igual a 1 si A es 1 o si $B = 0$ y $C = 1$. En otro caso D es igual a 0.

Se necesitan varias aclaraciones en relación con la notación. En ausencia de paréntesis, la operación AND es preferente a la operación OR. Además, cuando no hay ambigüedad, la operación AND se representa con una simple concatenación en lugar de con el operador punto. Por tanto,

Tabla B.1. Operaciones booleanas.

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P NAND Q	P NOR Q
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

$$A + B \cdot C = A + (B \cdot C) = A + BC$$

lo que quiere decir: hacer AND con B y C; luego hacer la OR con el resultado y A.

La Tabla B.1 define las operaciones lógicas básicas en una forma conocida como *tabla verdad*, que simplemente enumera el valor de una operación para cada combinación posible de los valores de los operandos. La tabla también enumera otros tres operadores útiles: XOR, NAND y NOR. La *exclusive-or* (XOR) de dos operandos lógicos es 1 si y solo si, uno de los operandos vale 1. La función NAND es el complemento (NOT) de la función AND, y la NOR es el complemento de la OR:

$$A \text{ NAND } B = \text{NOT}(A \text{ AND } B) = \overline{AB}$$
$$A \text{ NOR } B = \text{NOT}(A \text{ OR } B) = \overline{A + B}$$

Como veremos, estas tres operaciones nuevas pueden ser útiles para implementar ciertos circuitos digitales.

La Tabla B.2 resume las identidades clave del álgebra de Boole. Las ecuaciones se han organizado en dos columnas para mostrar la complementariedad, o dualidad, propias de las operaciones AND

Tabla B.2. Identidades básicas del álgebra de Boole.

Postulados básicos		
$A \cdot B = B \cdot A$	$A + B = B + A$	Conmutativa
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C)$ 5 $(A + B) \cdot (A + C)$	Distributiva
$1 \cdot A = A$	$0 + A = A$	Identidad
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$	Complemento
Otras identidades		
$0 \cdot A = 0$	$1 + A = 1$	
$A \cdot A = A$	$A + A = A$	
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$	Asociativa
$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$	Teorema de DeMorgan

y OR. Hay dos clases de identidades: las reglas básicas (o *postulados*) que se afirman sin demostración, y otras identidades que se pueden derivar de los postulados básicos. Los postulados definen la manera en la que expresiones booleanas se interpretan. Una de las dos leyes distributivas merece ser destacada ya que difiere de lo que encontraríamos en un álgebra normal:

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

Las dos últimas expresiones, se denominan teorema de DeMorgan. Se pueden reescribir de la siguiente forma:

$$A \text{ NOR } B = \overline{A} \text{ AND } \overline{B}$$
$$A \text{ NAND } B = \overline{A} \text{ OR } \overline{B}$$

Se invita al lector a verificar las expresiones de la Tabla B.2 sustituyendo las variables A, B y C por valores reales (unos y ceros).

B.2. PUERTAS

El bloque fundamental de construcción de todos los circuitos lógicos digitales son las puertas. Las funciones lógicas se implementan interconectando puertas.

Una puerta es un circuito electrónico que produce como señal de salida una operación booleana sencilla de las señales de entrada. Las puertas básicas usadas en lógica digital son AND, OR, NOT, NAND y NOR. La Figura B.1 muestra estas cinco puertas. Cada puerta se define de tres formas: símbolo gráfico, notación algebraica y tabla verdad. La simbología usada aquí y a lo largo del apéndice es el estándar IEEE, IEEE Std 91 [IEEE84]. Hay que destacar que la operación de inversión (NOT) se denota por un círculo.

Cada puerta tiene una o dos entradas y una salida. Cuando los valores de entrada cambian, la señal de salida correcta aparece casi instantáneamente, retrasada solo por el tiempo de propagación de la señal a través de la puerta (conocido como *retardo de puerta*). El significado de esto se verá en la Sección B.3.

Además de las puertas indicadas en la Figura B.1, se pueden usar puertas con tres, cuatro o más entradas. Por tanto, se puede implementar $X + Y + Z$ con una simple puerta OR de tres entradas.

Normalmente, no se usan todos los tipos de puertas en implementación. El diseño y la fabricación pueden ser más sencillos si solo se usan uno o dos tipos de puertas. Por tanto, es importante identificar conjuntos de puertas *funcionalmente completos*. Esto significa que cualquier función booleana se puede implementar usando solo las puertas del conjunto. Los siguientes conjuntos son funcionalmente completos:

- AND, OR, NOT
- AND, NOT
- OR, NOT
- NAND
- NOR

Nombre	Símbolo gráfico	Función algebraica	Tabla verdad
AND		$F = A \bullet B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND		$F = (\overline{AB})$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (\overline{A + B})$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0

Figura B.1. Puertas lógicas básicas.

Debería quedar claro que las puertas AND, OR y NOT constituyen un conjunto funcionalmente completo, ya que representan tres operaciones del álgebra de Boole. Para que las puertas AND y NOT formen un conjunto completo, debe haber una forma de sintetizar la operación OR a partir de las operaciones AND y NOT. Esto se puede hacer aplicando el teorema de DeMorgan:

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

$$\text{A OR B} = \text{NOT((NOT A)AND(NOT B))}$$

De igual forma, las operaciones OR y NOT son funcionalmente completas porque se pueden usar para sintetizar la operación AND.

La Figura B.2 muestra cómo se pueden implementar las funciones AND, OR y NOT únicamente con puertas NAND, y la Figura B.3 muestra lo mismo para NOR. Por esta razón, se pueden implementar circuitos digitales únicamente con puertas NAND o NOR, como frecuentemente se hace.

Con las puertas, se alcanza el nivel más primitivo de la ciencia e ingeniería de computadores. Un examen de las combinaciones de transistores usadas para construir puertas sale de este mundo para entrar en el mundo de la ingeniería electrónica. Para nuestros propósitos, sin embargo, nos es

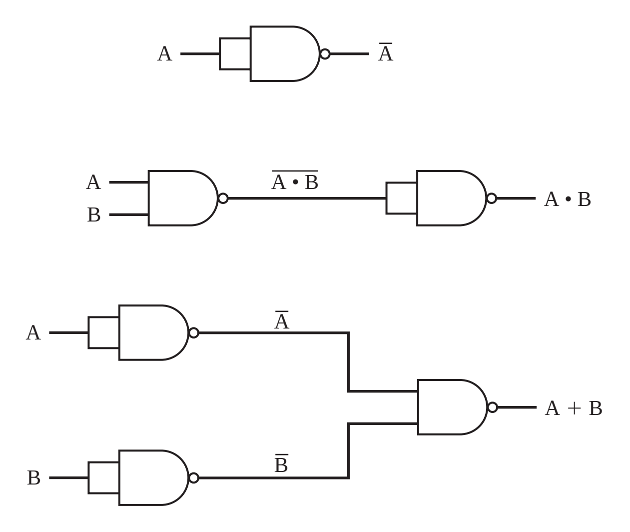


Figura B.2. Uso de las puertas NAND.

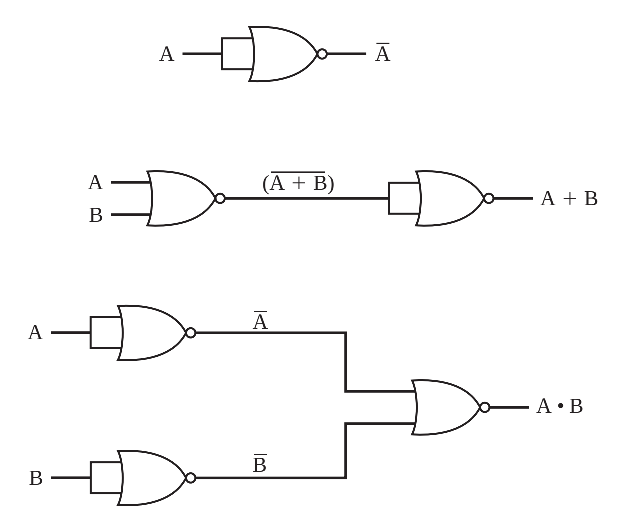


Figura B.3. Uso de las puertas NOR.

suficiente describir cómo se pueden usar las puertas como bloques de construcción para implementar los circuitos lógicos esenciales de un computador digital.

B.3. CIRCUITOS COMBINACIONALES

Un circuito combinacional es un conjunto de puertas interconectadas cuya salida, en un momento dado, es función solamente de la entrada en ese instante. Como ocurre con una puerta sencilla, la aparición de la entrada viene seguida casi inmediatamente por la aparición de la salida, con solo retardos de puerta.

En general, un circuito combinacional consiste en n entradas binarias y m salidas binarias. Como una puerta, un circuito combinacional puede definirse de tres formas:

- **Tabla verdad**: para cada una de las 2^n combinaciones posibles de las n señales de entrada, se enumera el valor binario de cada una de las m señales de salida.
- **Símbolo gráfico:** describe la organización de las interconexiones entre puertas.
- **Ecuaciones booleanas:** cada señal de salida se expresa como una función booleana de las señales de entrada.

IMPLEMENTACIÓN DE LAS FUNCIONES BOOLEANAS

Cualquier función booleana se puede implementar en electrónica en forma de red de puertas. Para una función dada, hay una serie de realizaciones alternativas. Considérese la función booleana representada por la tabla verdad de la Tabla B.3. Podemos expresar esta función sencillamente detallando las combinaciones de los valores de A, B, y C que hacen que F valga 1:

$$F = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C} \tag{B.1}$$

Hay tres combinaciones de los valores de entrada que hacen que F valga 1, y si se da cualquiera de estas tres combinaciones, el resultado será 1. Este tipo de expresión, por razones evidentes, se conoce como la forma *suma de productos* (SOP, *sum of products*). La Figura B.4, muestra una sencilla implementación con puertas AND, OR y NOT. Se puede obtener también otra forma de la tabla verdad. La forma SOP indica que la salida es 1 si cualquiera de las combinaciones de entrada que producen 1 es cierta. También se puede decir que la salida es 1 si ninguna de las combinaciones de entrada que producen 0 es cierta. Por tanto:

$$F = \overline{(\overline{A}\,\overline{B}\,\overline{C})} \cdot \overline{(\overline{A}\,\overline{B}\,C)} \cdot \overline{(A\,\overline{B}\,\overline{C})} \cdot \overline{(A\,\overline{B}\,C)} \cdot \overline{(A\,B\,C)}$$

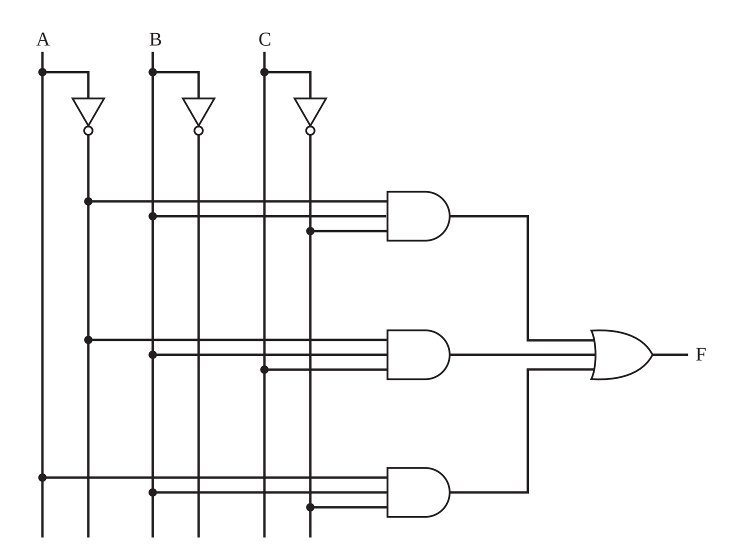


Figura B.4. Implementación de la suma de productos de la Tabla B.3.

Esta expresión se puede reescribir usando una generalización del teorema de DeMorgan:

$$\overline{(X \cdot Y \cdot Z)} = \overline{X} + \overline{Y} + \overline{Z}$$

Por tanto,

$$\begin{aligned} F &= (\overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}}) \cdot (\overline{\overline{A}} + \overline{\overline{B}} + \overline{C}) \cdot (\overline{A} + \overline{\overline{B}} + \overline{\overline{C}}) \cdot (\overline{A} + \overline{\overline{B}} + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) \\ &= (A + B + C) \cdot (A + B + \overline{C}) \cdot (\overline{A} + B + C) \cdot (\overline{A} + B + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) \qquad \text{(B.2)} \end{aligned}$$

Esta última expresión está en la forma de *producto de sumas* (POS, *product of sums*), como se ilustra en la Figura B.5. Por claridad, no aparecen las puertas NOT. En su lugar, suponemos que se dispone de cada señal de entrada y de su complemento. Esto simplifica el diagrama lógico y hace más legibles las entradas a las puertas.

Por tanto, se puede realizar una función booleana tanto en la forma SOP como en la forma POS. En este momento, podría parecer que la elección dependería de si la tabla verdad contiene más unos o ceros para la función de salida: la SOP tiene un término para cada 1, y la POS tiene un término para cada 0. Sin embargo, hay otras consideraciones:

- Generalmente es posible obtener una expresión booleana más sencilla de la tabla verdad que de las formas SOP o POS.
- Puede ser preferible implementar la función con puertas sencillas (NAND o NOR).

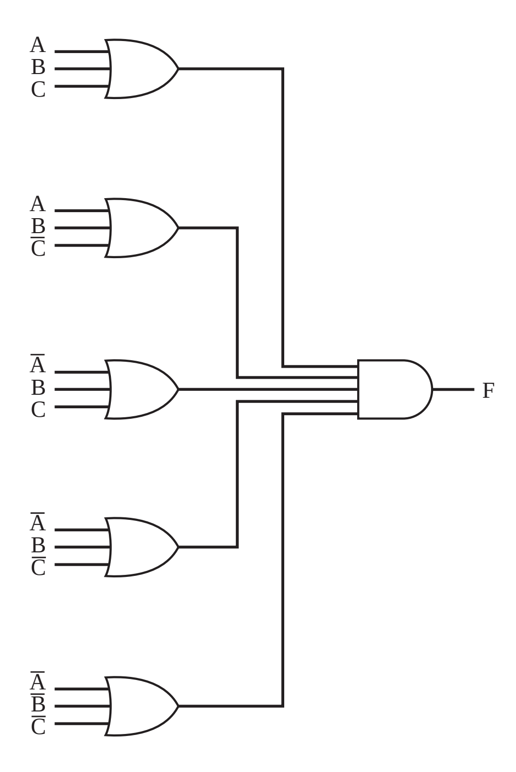


Figura B.5. Implementación del producto de sumas.

El significado del primer punto es que, con una expresión booleana más sencilla, se necesitan menos puertas para implementar la función. Para llevar a cabo esta simplificación se pueden usar tres métodos:

- Simplificación algebraica
- Mapas de Karnaugh
- Tablas de Quine-McKluskey

Simplificación algebraica. La simplificación algebraica supone la aplicación de las identidades de la Tabla B.3, que reduce la expresión booleana a otra con menos elementos.

Por ejemplo, supongamos de nuevo la Ecuación B.1. Un poco de razonamiento debería convencer al lector de que una expresión equivalente es

$$F = \overline{A}B + B\overline{C} \qquad \text{(B.3)}$$

O, incluso más sencillamente,

$$F = B(\overline{A} + \overline{C})$$

Esta expresión se puede implementar como se indica en la Figura B.6. La simplificación de la Ecuación B.1 se ha hecho esencialmente por observación. Para obtener una expresión más compleja, se necesita un procedimiento más sistemático.

Mapas de Karnaugh. Si se quiere simplificar, los mapas de Karnaugh son una forma conveniente de representar una función booleana con pocas variables (de cuatro a seis). El mapa es un conjunto de 2^n cuadrículas, que representan las posibles combinaciones de los valores de *n* variables binarias. La Figura B.7a muestra el mapa de cuatro cuadrículas para una función de dos variables. Es conveniente, para futuros propósitos, enumerar las combinaciones en el orden 00, 01, 11, 10. Como las cuadrículas corresponden a combinaciones que se van a usar para escribir información, las combinaciones se escriben habitualmente externamente, en la parte superior de las cuadrículas. En el caso de tres variables, la representación es un conjunto de ocho cuadrículas (Figura B.7b), con los valores de una de las variables a la izquierda y para las otras dos variables encima de las cuadrículas. Para cuatro variables, se necesitan 16 cuadrículas, con la disposición indicada en la Figura B.7c.

El mapa se puede usar para representar cualquier función booleana de la siguiente forma. Cada cuadrícula corresponde a un único producto en la forma de suma de productos, con valor 1 correspondiente a la variable y valor 0 correspondiente a la NOT de dicha variable. Por tanto, el producto

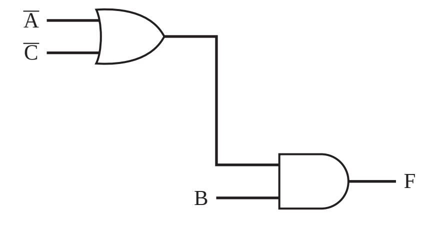


Figura B.6. Implementación simplificada de la Tabla B.3.

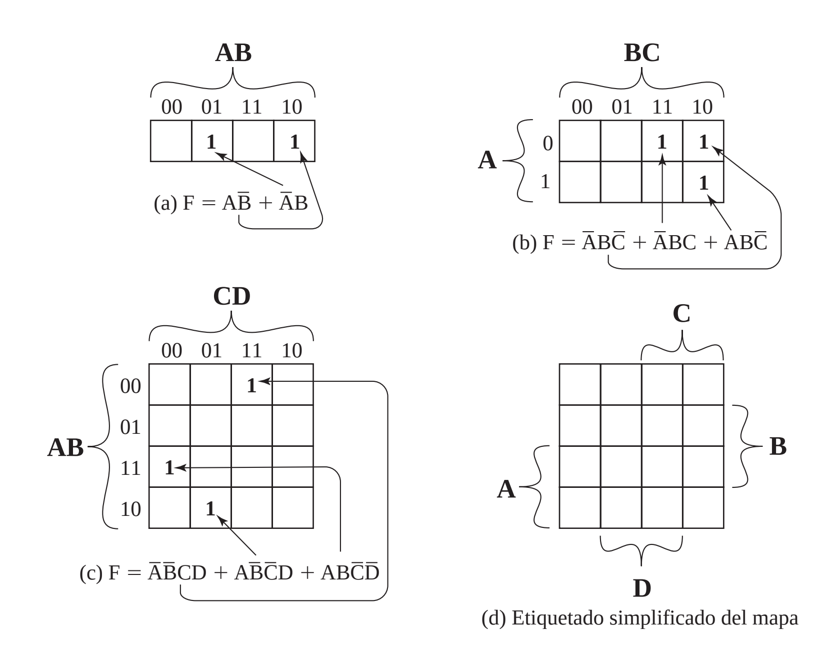


Figura B.7. La utilización de los mapas de Karnaugh para representar funciones booleanas.

A$\overline{B}$ corresponde a la cuarta cuadrícula de la Figura B.7a. Para cada uno de estos productos de la función, se coloca un 1 en la cuadrícula correspondiente. Por tanto, para el ejemplo de dos variables, el mapa corresponde a A$\overline{B}$ + $\overline{A}$B. Dada la tabla verdad de una función booleana, es fácil construir el mapa: para cada combinación de los valores de las variables que dan como resultado 1 en la tabla verdad, se pone un 1 en la cuadrícula correspondiente. La Figura B.7b muestra el resultado para la tabla verdad de la Tabla B.3. Para pasar de una expresión booleana a un mapa, primero es necesario poner la expresión en lo que se denomina forma *canónica*: cada término de la expresión debe contener cada variable. Así, por ejemplo, si se tiene la Ecuación A-3, debemos expandirla primero a la forma completa de la Ecuación A-1 y después pasarla al mapa.

Los rótulos usados en la Figura B.7d enfatizan la relación entre las variables y las filas y columnas del mapa. Aquí, las dos filas que abarca el símbolo A son aquellas en las que la variable

Tabla B.3. Función booleana de tres variables.

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

A vale 1; las filas que no abarca el símbolo A son aquellas en las que A vale 0 (lo mismo ocurre para B, C, y D).

Una vez que se ha creado el mapa de una función, podemos escribir, a menudo, una expresión algebraica sencilla anotando el conjunto de unos del mapa. El principio es el siguiente. Dos casillas adyacentes cualesquiera difieren en solo una de las variables. Si dos casillas adyacentes contienen un 1, entonces los correspondientes términos producto difieren solo en una variable. En tal caso, los dos términos se pueden fundir en uno eliminando esta variable. Por ejemplo, en la Figura B.8a, las dos casillas adyacentes corresponden a los términos $\overline{A}B\overline{C}D$ y $\overline{A}BCD$. Por tanto, la función se puede expresar así:

$$\overline{A}B\overline{C}D + \overline{A}BCD = \overline{A}BD$$

Este proceso se puede ampliar de varias formas. Primero, el concepto de adyacencia se puede ampliar para incluir el recubrimiento alrededor del borde del mapa. Por tanto, la casilla más alta de

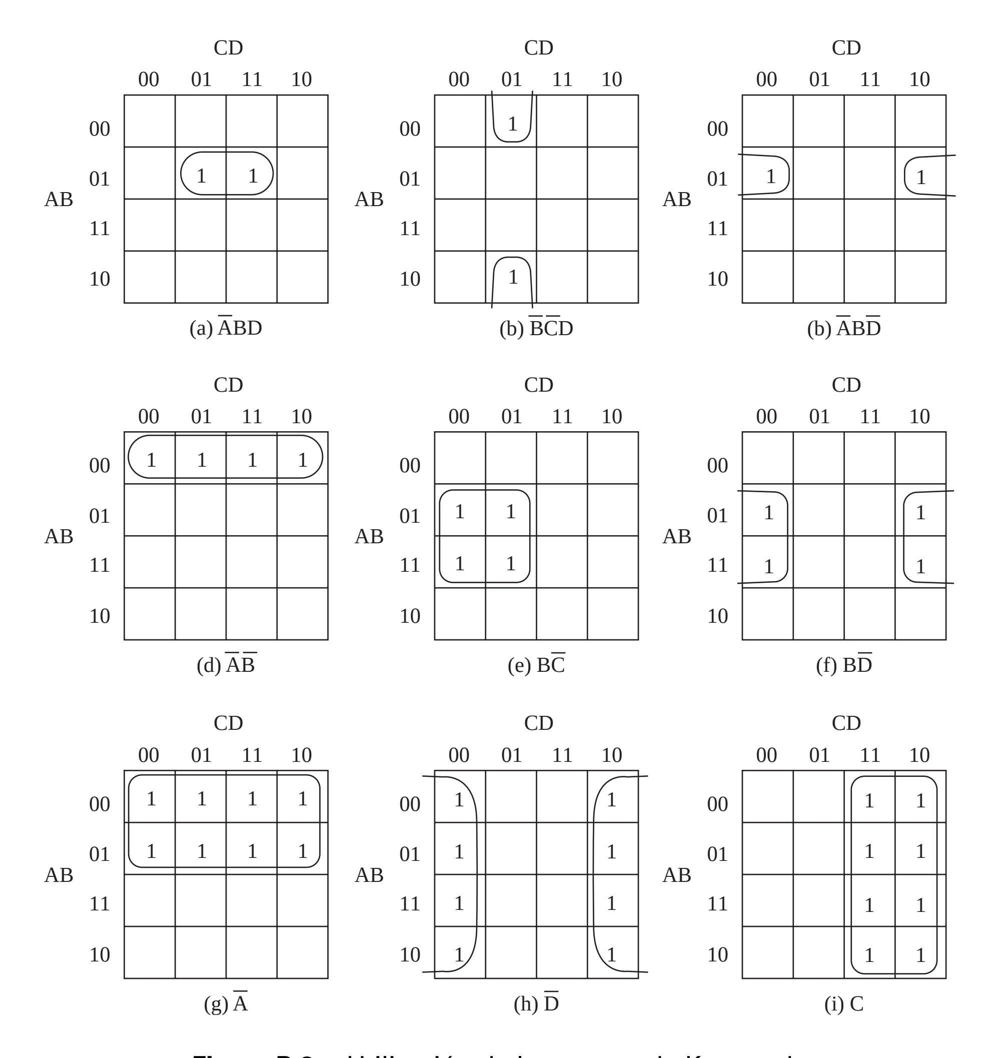


Figura B.8. Utilización de los mapas de Karnaugh.

una columna es adyacente a la más baja, y la casilla más a la izquierda de la fila es adyacente a la que está más a la derecha. Estas condiciones se ilustran en las Figuras B. 8b y c. Segundo, podemos agrupar no solo dos casillas, sino 2^n casillas adyacentes, es decir, 4, 8, etc. Los tres siguientes ejemplos de la Figura B.8, muestran agrupaciones de cuatro casillas. Hay que destacar que en este caso, se pueden eliminar dos de las variables. Los tres últimos ejemplos muestran grupos de ocho cuadrículas, que permiten eliminar tres variables.

Para simplificar podemos resumir las reglas como sigue:

1. Entre las casillas marcadas (casillas con un 1), encontrar aquellas que pertenezcan a un único bloque lo mayor posible ya sea de 1, 2, 4 u 8 casillas, y rodear el bloque con un círculo.
2. Seleccionar bloques adicionales de casillas marcadas tan grandes y tan pocas como sea posible, pero que incluyan a cada casilla marcada al menos una vez. Los resultados pueden no ser únicos en algunos casos. Por ejemplo, si una casilla marcada se combina exactamente con otras dos, y no hay una cuarta para completar un grupo mayor, entonces se puede elegir entre dos agrupaciones. Cuando se seleccionan grupos se permite usar el mismo uno más de una vez.
3. Seguir dibujando círculos alrededor de las casillas aisladas marcadas, o de parejas de casillas marcadas adyacentes, o de grupos de cuatro, ocho, etc. de forma que cada cuadrado marcado pertenezca al menos a un círculo; luego utilizar el menor número posible de bloques para incluir a todas las casillas marcadas.

La Figura B.9a, basada en la Tabla B.3, ilustra el procedimiento. Si queda algún 1 aislado después de haber agrupado, entonces, cada uno de ellos se rodea con un círculo como si fuera un grupo de unos. Finalmente, antes de pasar el mapa a una expresión simplificada Booleana, cualquier grupo de unos que esté completamente solapado por otros grupos se puede eliminar. Es lo que se muestra

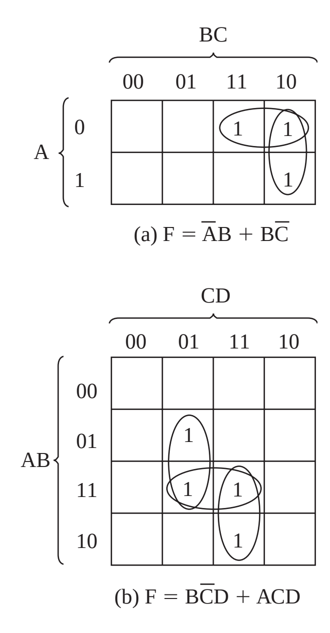


Figura B.9. Grupos solapados.

en la Figura B.9b. En este caso el grupo horizontal es redundante y se puede ignorar a la hora de crear la expresión booleana.

Es necesario mencionar una característica adicional de los mapas de Karnaugh. En algunos casos, ciertas combinaciones de valores de las variables no se dan nunca, y por consiguiente, la salida correspondiente no se produce tampoco. Estas se denominan condiciones de «indiferencia». Para cada una de estas condiciones, se coloca la letra «d» en la casilla correspondiente del mapa. Cuando se hace la agrupación y simplificación, cada «d» puede tratarse como un 1 o un 0, eligiendo lo que conduzca a una expresión más sencilla.

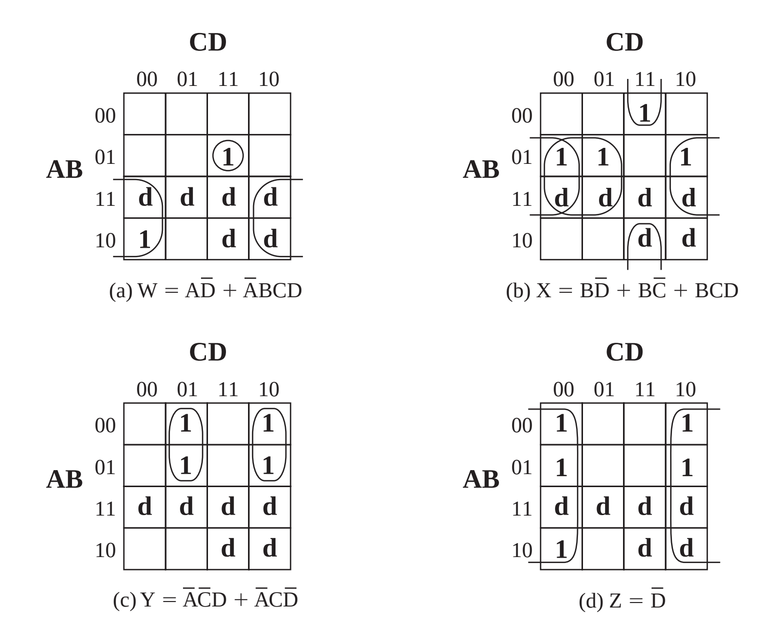
Un ejemplo, presentado en [HAYE88], ilustra lo que hemos estado discutiendo. Nos gustaría desarrollar las expresiones booleanas para un circuito que suma un 1 a los dígitos decimales empaquetados. Recordemos de la Sección 9.2 que con decimales empaquetados, cada dígito decimal se representa con un código de cuatro bits, de una forma obvia.

Así, 0 = 0000, 1 = 0001, ..., 8 = 1000, y 9 = 1001. Las combinaciones de cuatro bits restantes, de 1010 a 1111, no se usan. Este código también se denomina Decimal Codificado en Binario (BCD, *Binary Coded Decimal*).

La Tabla B.4 muestra la tabla verdad para producir un resultado de cuatro bits que es la entrada BCD de cuatro bits incrementada en 1. La suma es en módulo 10. Así, 9 + 1 = 0. También, hay que notar que seis de los códigos de entrada producen «indiferencias» como resultado, ya que esas no son entradas BCD válidas. La Figura B.10 muestra el resultado de los mapas de Karnaugh para cada una de las variables de salida. Las casillas «d» se usan para lograr las mejores agrupaciones posibles.

Tabla B.4. Tabla verdad de un contador digital de un dígito.

Número	Entrada				Número	Salida			
	A	B	C	D		W	X	Y	Z
0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	2	0	0	1	0
2	0	0	1	0	3	0	0	1	1
3	0	0	1	1	4	0	1	0	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	6	0	1	1	0
6	0	1	1	0	7	0	1	1	1
7	0	1	1	1	8	1	0	0	0
8	1	0	0	0	9	1	0	0	1
9	1	0	0	1	0	0	0	0	0
Indiferencias	1	0	1	0		d	d	d	d
	1	0	1	1		d	d	d	d
	1	1	0	0		d	d	d	d
	1	1	0	1		d	d	d	d
	1	1	1	0		d	d	d	d
	1	1	1	1		d	d	d	d



(a) $W = A\overline{D} + \overline{A}BCD$

(b) $X = B\overline{D} + B\overline{C} + BCD$

(c) $Y = \overline{A}\overline{C}D + \overline{A}C\overline{D}$

(d) $Z = \overline{D}$

Figura B.10. Mapas de Karnaugh para el ejemplo del contador.

El método de Quine-McKluskey. Cuando se incrementa en más de cuatro variables, el método del mapa de Karnaugh se va haciendo cada vez más incómodo. Con cinco variables, se necesitan dos mapas de 16 × 16, con un mapa situado encima del otro, en tres dimensiones, para conseguir la adyacencia. ¡Seis variables requieren cuatro tablas de 16 × 16 en cuatro dimensiones! Un procedimiento alternativo es una técnica tabular, denominada método de Quine-McKluskey. Este método es adecuado para programar en un computador y así tener una herramienta automática que produzca expresiones booleanas minimizadas.

Este método se explica mejor mediante un ejemplo. Consideremos la siguiente expresión:

$$ABCD + AB\overline{C}D + AB\overline{C}\,\overline{D} + A\overline{B}CD + \overline{A}BCD + \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}D$$

Supongamos que esta expresión se ha obtenido de una tabla verdad. Nos gustaría conseguir una expresión mínima adecuada para implementarla con puertas.

El primer paso es construir una tabla en la que cada fila corresponda a un término producto de la expresión. Los términos se agrupan de acuerdo con el número de variables complementadas. Es decir, empezamos con el termino sin complementos, si existe, luego todos los términos con un complemento, etc. La Tabla B.5 muestra la lista para nuestra expresión ejemplo, indicando con las líneas horizontales las agrupaciones. Para más claridad, cada término se representa con un 1 para cada variable sin complementar y con un 0 para cada variable complementada. Por tanto, agrupamos términos de acuerdo con el número de unos que contiene. La columna «índice» es simplemente el equivalente decimal y es útil para lo que se explicará más adelante.

El siguiente paso es encontrar todas las parejas de términos que difieren en solo una variable, es decir, todos los pares de términos que son iguales excepto que, una variable es 0 en uno de los términos y 1 en el otro. Debido a la manera en la que se agrupan términos, podemos hacerlo empezando

Tabla B.5. Primer paso del método de Quine-McKluskey
(para $ABCD + AB\overline{C}D + AB\overline{C}\,\overline{D} + A\overline{B}CD + \overline{A}BCD + \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}D$).

Término producto	**Índice**	**A**	**B**	**C**	**D**	
$\overline{A}\,\overline{B}\,\overline{C}D$	1	0	0	0	1	✓
$\overline{A}B\overline{C}D$	5	0	1	0	1	✓
$\overline{A}BC\overline{D}$	6	0	1	1	0	✓
$AB\overline{C}\,\overline{D}$	12	1	1	0	0	✓
$\overline{A}BCD$	7	0	1	1	1	✓
$A\overline{B}CD$	11	1	0	1	1	✓
$AB\overline{C}D$	13	1	1	0	1	✓
$ABCD$	15	1	1	1	1	✓

con el primer grupo y comparar cada término del primer grupo con cada término del segundo grupo. Después comparamos cada término del segundo grupo con todos los términos del tercer grupo, y así sucesivamente. Cuando se encuentra un emparejamiento, se coloca una marca en cada término, se combina el par eliminando la variable en la que difieren los dos términos y se añade a la nueva lista. Así, por ejemplo, los términos $\overline{A}BC\overline{D}$ y $\overline{A}BCD$ se combinan para producir el término ABC. Este proceso continúa hasta que se haya analizado la tabla original entera. El resultado es una nueva tabla con los siguientes elementos:

$\overline{A}CD$	$AB\overline{C}$	$ABD\surd$
	$B\overline{C}D\surd$	ACD
	$\overline{A}BC$	$BCD\surd$
	$\overline{A}BD\surd$	

La nueva tabla se organiza en grupos, como indicamos antes, de la misma forma que la primera tabla. La segunda tabla se procesa, después, de la misma manera que la primera. Es decir, se comprueban términos que difieren en solo una variable y se produce un nuevo término para una tercera tabla. En este ejemplo, la tercera tabla que se hace contiene solamente un término: BD.

En general, el proceso continuaría a través de sucesivas tablas hasta una tabla en la que no haya emparejamientos. En este caso, hay implicadas tres tablas.

Una vez se haya completado el proceso descrito anteriormente, tenemos que eliminar muchos de los posibles términos de la expresión. Aquellos términos que no hayan sido eliminados se usan para construir una matriz, como se ilustra en la Tabla B.6. Cada fila de la matriz corresponde a uno de los términos que no se han eliminado (no tiene marca) en cualquiera de las tablas usadas anteriormente. Cada columna se corresponde con uno de los términos de la expresión original. Se coloca una X en cada intersección de una fila y una columna tal que el elemento de la fila sea “compatible” con el elemento de la columna. Es decir, las variables presentes en el elemento de la fila tienen el mismo valor que las variables presentes en el elemento de la columna. Después, se rodea con un círculo cada X que esté sola en una columna. Entonces, se sitúa un cuadrado alrededor de cada X en cualquier fila

Tabla B.6. Último paso del método de Quine-McKluskey (para $F = ABCD + AB\overline{C}D + AB\overline{C}\overline{D} + A\overline{B}CD + \overline{A}BCD + \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + \overline{A}\overline{B}\overline{C}D$).

	$ABCD$	$AB\overline{C}D$	$AB\overline{C}\overline{D}$	$A\overline{B}CD$	$\overline{A}BCD$	$\overline{A}BC\overline{D}$	$\overline{A}B\overline{C}D$	$\overline{A}\overline{B}\overline{C}D$
BD	X	X			X		X	
$\overline{A}\overline{C}D$							$\boxed{X}$	$\otimes$
$\overline{A}BC$					$\boxed{X}$	$\otimes$		
$AB\overline{C}$		$\boxed{X}$	$\otimes$					
ACD	$\boxed{X}$			$\otimes$				

que tenga un círculo en la X. Si cada columna tiene ahora una X encerrada en un cuadrado o en un círculo, entonces ya se ha concluido, y los elementos de esa fila cuyas X estén marcadas constituyen la expresión mínima. Por tanto, en nuestro ejemplo, la expresión final es:

$$AB\overline{C} + ACD + \overline{A}BC + \overline{A}\overline{C}D$$

En los casos en los que algunas columnas no tengan ni un círculo ni un cuadrado, se necesita un proceso adicional. Esencialmente, seguimos añadiendo elementos a la expresión hasta que se cubran todas las columnas.

Resumamos el método Quine-McKluskey para intentar justificar intuitivamente cómo se realiza la minimización. La primera fase de la operación es razonablemente sencilla. El proceso elimina variables innecesarias en términos producto. Entonces, la expresión $AB\overline{C} + ABC$ es equivalente a AB, dado que

$$ABC + AB\overline{C} = AB(C + \overline{C}) \text{ 5 } AB$$

Tras la eliminación de las variables, nos queda una expresión que es claramente equivalente a la expresión original. Sin embargo, puede haber términos redundantes en la expresión, como encontramos agrupaciones redundantes en los mapas de Karnaugh. La organización de la matriz asegura que se incluye (cubre) cada término de la expresión original y lo hace de forma que minimiza el número de términos en la expresión final.

Implementaciones NAND y NOR. Otra consideración en la implementación de funciones booleanas concierne a los tipos de puertas usados. Es a menudo deseable implementar una función booleana solo con puertas NAND o solo con puertas NOR. Aunque pueda no ser la implementación con un mínimo de puertas, tiene la ventaja de la regularidad, que puede simplificar el proceso de fabricación. Consideremos de nuevo la Ecuación B.3:

$$F = B(\overline{A} + \overline{C})$$

Como el complemento del complemento es el valor original,

$$F = B(\overline{A} + \overline{C}) = \overline{\overline{\overline{(\overline{A}B)} + \overline{(B\overline{C})}}}$$

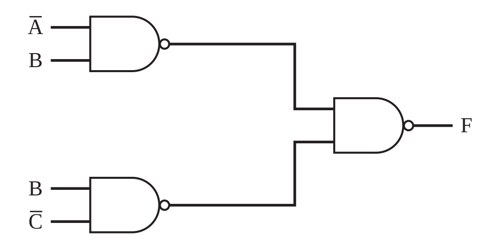


Figura B.11. Implementación con NAND de la Tabla B.3.

aplicando el teorema de DeMorgan,

$$F = \overline{\overline{(\overline{A}B)} \cdot \overline{(B\overline{C})}}$$

que tiene tres operadores NAND, como se ilustra en la Figura B.11.

MULTIPLEXORES

El multiplexor conecta varias entradas con una única salida. En un momento dado, se selecciona una de las entradas para que pase a la salida. La Figura B.12 muestra una representación en diagrama de bloques general.

Representa un multiplexor de 4 a 1. Hay cuatro líneas de entrada, llamadas D0, D1, D2 y D3. Se selecciona una de estas líneas para dar la señal de salida F. Para seleccionar una de las cuatro entradas posibles, se necesita un código de selección de dos bits, que se implementa con dos líneas de selección llamadas S1 y S2.

La tabla verdad de la Tabla B.7 define un ejemplo de un multiplexor de cuatro a uno. Es una forma simplificada de una tabla verdad. En vez de mostrar todas las combinaciones posibles de las variables de entrada, muestra la salida como dato procedente de la línea D0, D1, D2, o D3. La Figura B.13 muestra una implementación usando puertas AND, OR y NOT. S1 y S2 se conectan a las puertas AND de forma que, para cualquier combinación de S1 y S2, tres de las puertas AND tengan la salida a 0. La cuarta puerta AND sacará el valor de la línea seleccionada que será 0 o 1. Por tanto,

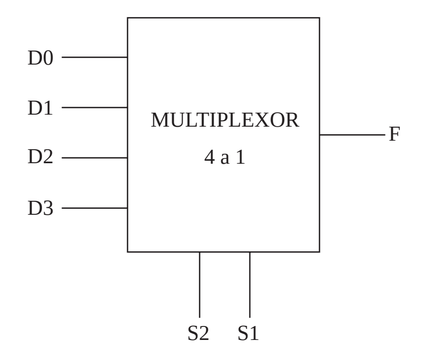


Figura B.12. Representación de multiplexor de 4 a 1.

Tabla B.7. Tabla verdad de un multiplexor 4 a 1.

S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

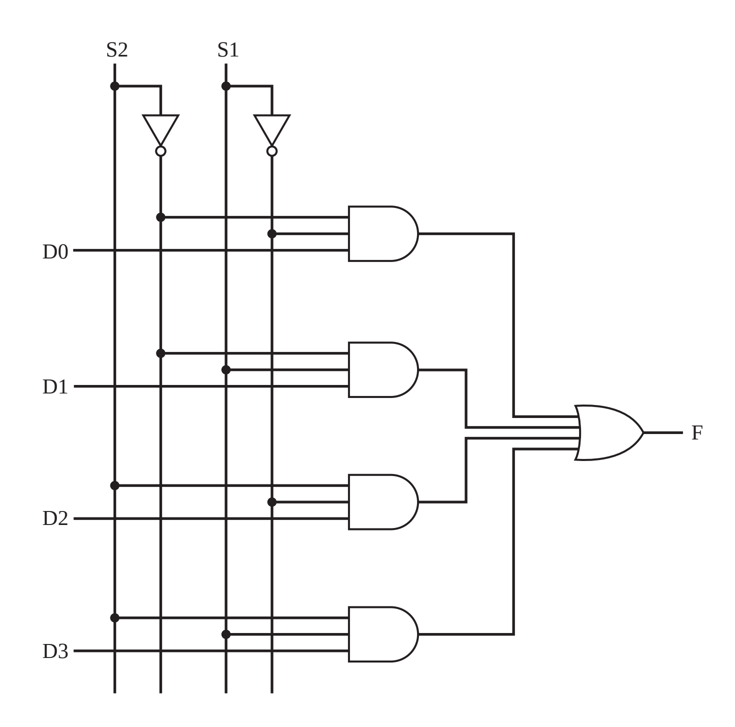


Figura B.13. Implementación de un multiplexor.

tres de las entradas de la puerta OR son siempre 0, y la salida de la puerta OR será igual al valor de la puerta de entrada seleccionada. Usando esta organización regular, es fácil construir multiplexores de ocho a uno, de 16 a uno, etc.

Los multiplexores se usan en circuitos digitales para controlar el enrutamiento de señales y datos. Un ejemplo es la carga del contador de programa (PC). El valor a cargar en el contador de programa puede venir de una o varias fuentes diferentes:

- De un contador binario, si el PC se va a incrementar para la siguiente instrucción.
- Del registro de instrucción, si se acaba de ejecutar una instrucción de salto usando direccionamiento directo.
- De la salida de la ALU, si la instrucción de salto especifica la dirección usando modo de desplazamiento.

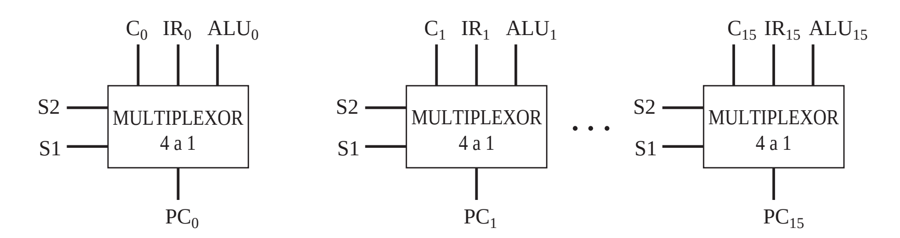


Figura B.14. Multiplexor de entrada al contador de programa.

Las distintas entradas se pueden conectar a las líneas de entrada de un multiplexor con el PC conectado a la línea de salida. Las líneas seleccionadas determinan cuál es el valor a cargar en el PC. Como el PC contiene varios bits, se usan varios multiplexores, uno por bit. La Figura B.14 ilustra esto para direcciones de 16 bits.

DECODIFICADORES

Un decodificador es un circuito combinacional con varias líneas de salida, con una sola de ellas seleccionada en un instante dado, dependiendo del patrón de líneas de entrada. En general, un decodificador tiene *n* entradas y 2^n salidas. La Figura B.15 muestra un decodificador con tres entradas y ocho salidas.

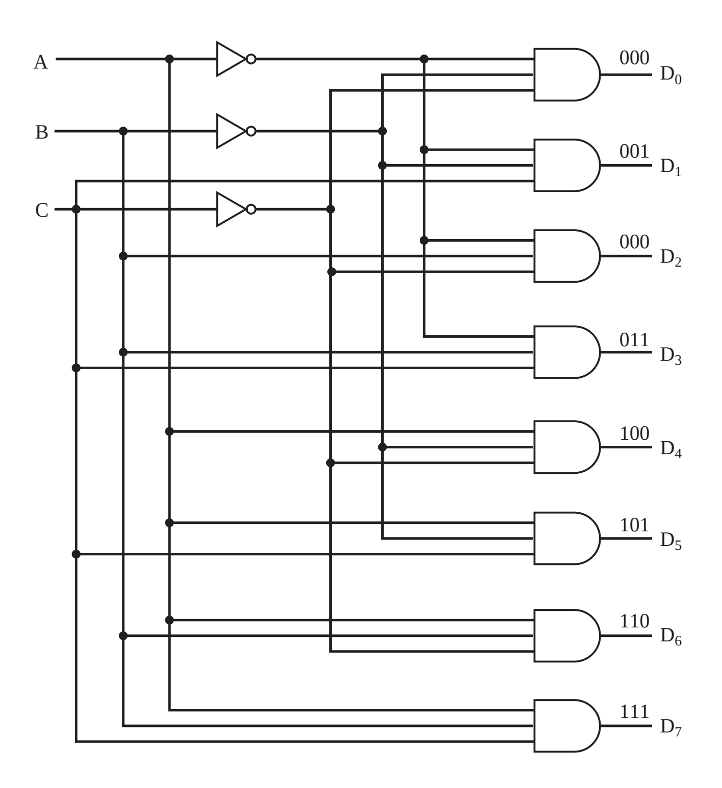


Figura B.15. Decodificador con tres entradas y $2^3 = 8$ salidas

Los decodificadores tienen muchos usos en computadores digitales. Un ejemplo es la decodificación de direcciones. Supongamos que queremos construir una memoria de 1 KB usando cuatro chips RAM de 256 × 8 bits. Queremos un espacio de direcciones único y unificado, que se pueda descomponer como sigue:

Dirección	Chip
0000-00FF	0
0100-01FF	1
0200-02FF	2
0300-03FF	3

Cada chip requiere ocho líneas de dirección, y estos se toman de los ocho bits menos significativos de la dirección. Los dos bits más significativos de los diez bits de dirección, se usan para seleccionar uno de los cuatro chips RAM. Para ello, se usa un decodificador de dos a cuatro cuya salida habilita uno de los cuatro chips, como se muestra en la Figura B.16.

Con una línea adicional de entrada, se puede usar el decodificador como demultiplexor. El demultiplexor realiza la función inversa de un multiplexor; conecta una única entrada a una o varias salidas. Esto se ve en la Figura B.17. Como antes, se decodifican *n* entradas para producir una única salida de las 2^n. Con todas las 2^n líneas de salida se hace la operación AND con un dato de la línea de entrada.

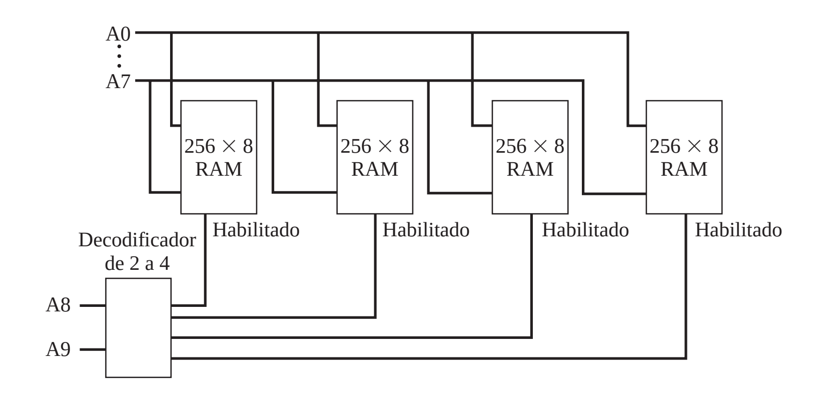


Figura B.16. Decodificación de una dirección.

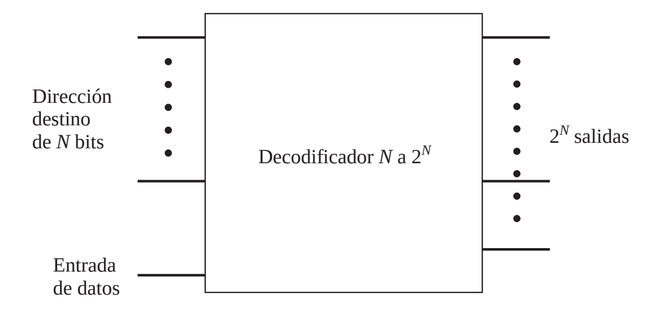


Figura B.17. Implementación de un demultiplexor usando un decodificador.

Por tanto, las *n* entradas actúan como una dirección para seleccionar una línea de salida concreta, y el valor del dato de la línea de entrada (0 o 1) se encamina a dicha línea de salida.

La configuración de la Figura B.17 se puede ver de otra forma. Cambiar el nombre de la nueva línea, *Entrada de datos,* por *Habilitar.* Esto permite el control de la temporización del decodificador. La salida decodificada aparece solo cuando está presente la entrada codificada *y* la línea de habilitación valga uno.

ARRAY LÓGICO PROGRAMABLE (PLA*, PROGRAMMABLE LOGIC ARRAY*)

Hasta ahora, hemos tratado las puertas individuales como bloques de construcción, para realizar funciones arbitrarias. El diseñador podría seguir una estrategia de minimización del número de puertas que van a usarse, manipulando las expresiones booleanas correspondientes.

Como el nivel de integración de los circuitos integrados aumenta, se tienen en cuenta otras consideraciones. Los primeros circuitos integrados, usando una escala de integración pequeña (SSI), contenían de una a diez puertas en un chip. Cada puerta se trata independientemente, en su uso como bloques de construcción descrito hasta ahora. La Figura B.18 es un ejemplo de algunos chips SSI. Para construir una función lógica, se disponen varios de estos chips en una tarjeta de circuito impreso y se hacen las interconexiones adecuadas entre los terminales de los chips.

El incremento del nivel de integración hace posible añadir más puertas en un chip y hacer interconexiones dentro del chip también. Esto tiene como ventajas la disminución del coste y del tamaño, y el incremento de velocidad (puesto los retardos dentro del chip son menores que los retardos fuera del chip). Sin embargo, surge un problema de diseño. Para cada función lógica particular o conjunto de funciones, hay que diseñar la organización de las puertas e interconexiones en el chip. El coste y el tiempo implicados en el diseño del chip a medida, son altos. Entonces, se convierte en algo atractivo el desarrollo de un chip de uso general que esté listo para adaptarse a usos específicos. Esta es la intención de los *arrays lógicos programables* (PLA, *programmable logic array*).

Los PLA se basan en el hecho de que cualquier función booleana (tabla verdad) se puede expresar en forma de suma de productos (SOP), como hemos visto. Un PLA cosiste en una disposición regular de puertas NOT, AND y OR en un chip. Cada entrada al chip pasa a través de una puerta NOT, así que cada entrada y su complemento están disponibles para cada puerta AND. La salida de cada puerta AND está disponible para cada puerta OR, y la salida de cada puerta OR es una salida del chip. Haciendo las conexiones adecuadas, se pueden implementar expresiones SOP arbitrarias.

La Figura B.19a muestra un PLA con tres entradas, ocho puertas, y dos salidas. Los PLAs mayores contienen varios cientos de puertas, de quince a 25 entradas, y de cinco a quince salidas. Las conexiones de las entradas a las puertas AND, y de las puertas AND a las puertas OR, no están especificadas.

Los PLAs se fabrican de dos formas diferentes para permitir una programación más fácil (hacer las conexiones). En la primera, cada conexión posible se hace a través de fusible en cada punto de intersección. Este tipo de PLA se denomina *array lógico programable de campo.* Alternativamente, las propias conexiones se pueden hacer durante la fabricación del chip usando una máscara adecuada dada para un patrón de interconexión particular. En cualquiera de los casos, la PLA proporciona una forma de implementación de funciones lógicas digitales flexible y barata.

En la Figura B.19b se muestra un diseño que sintetiza dos expresiones booleanas.

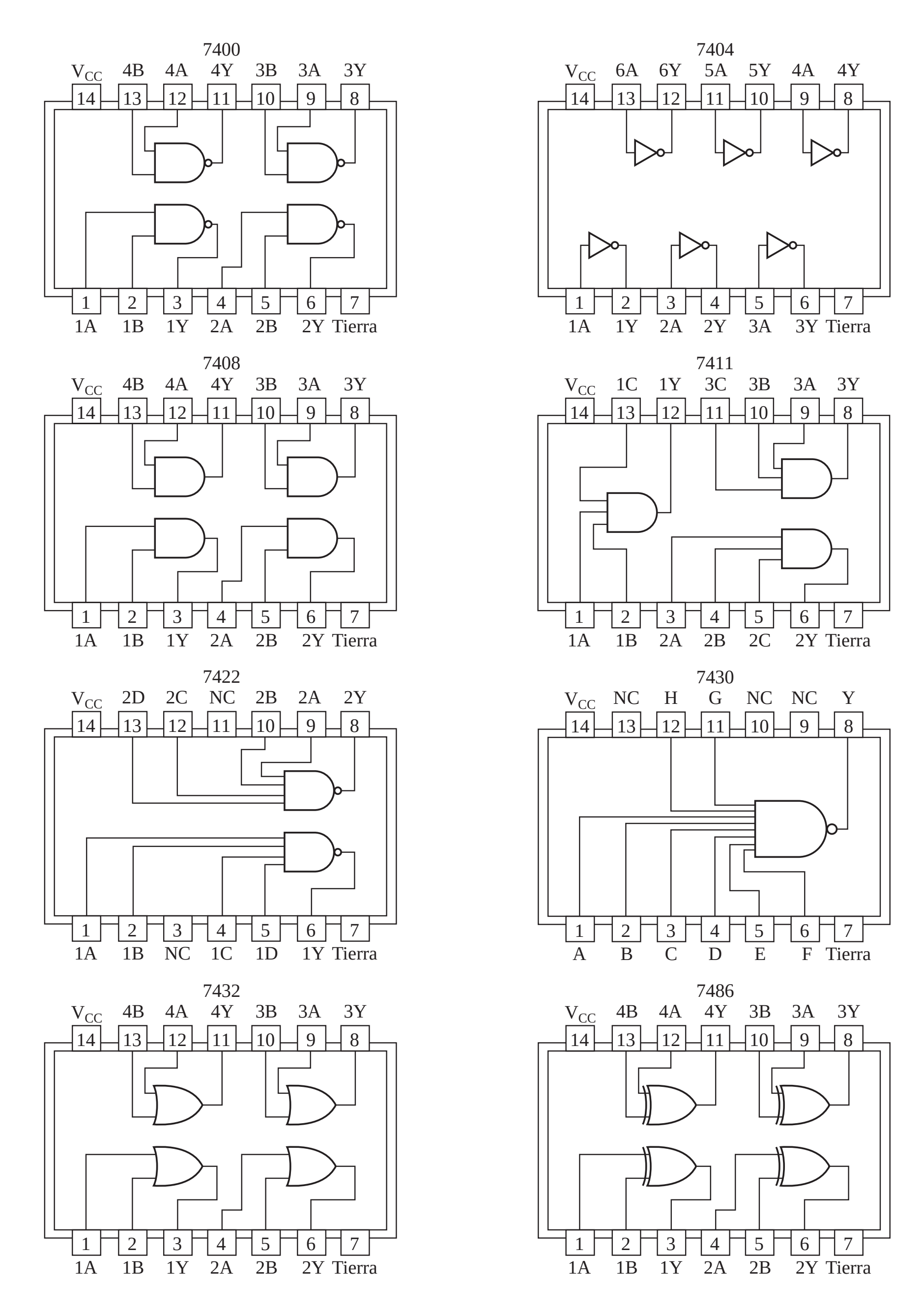


Figura B.18. Algunos chips SSi. Organización de los pines de *"The TTL data book for design engineers"*, copyright 1976 Texas Instruments Incorporated.

MEMORIA DE SOLO LECTURA (ROM, *READ ONLY MEMORY*)

A los circuitos combinacionales se les llama a veces circuitos «sin memoria», ya que su salida depende solo de la entrada actual y no retiene la historia de las entradas anteriores. Sin embargo, hay un tipo de memoria que se implementa con circuitos combinacionales, llamada *memoria de solo lectura* (ROM).

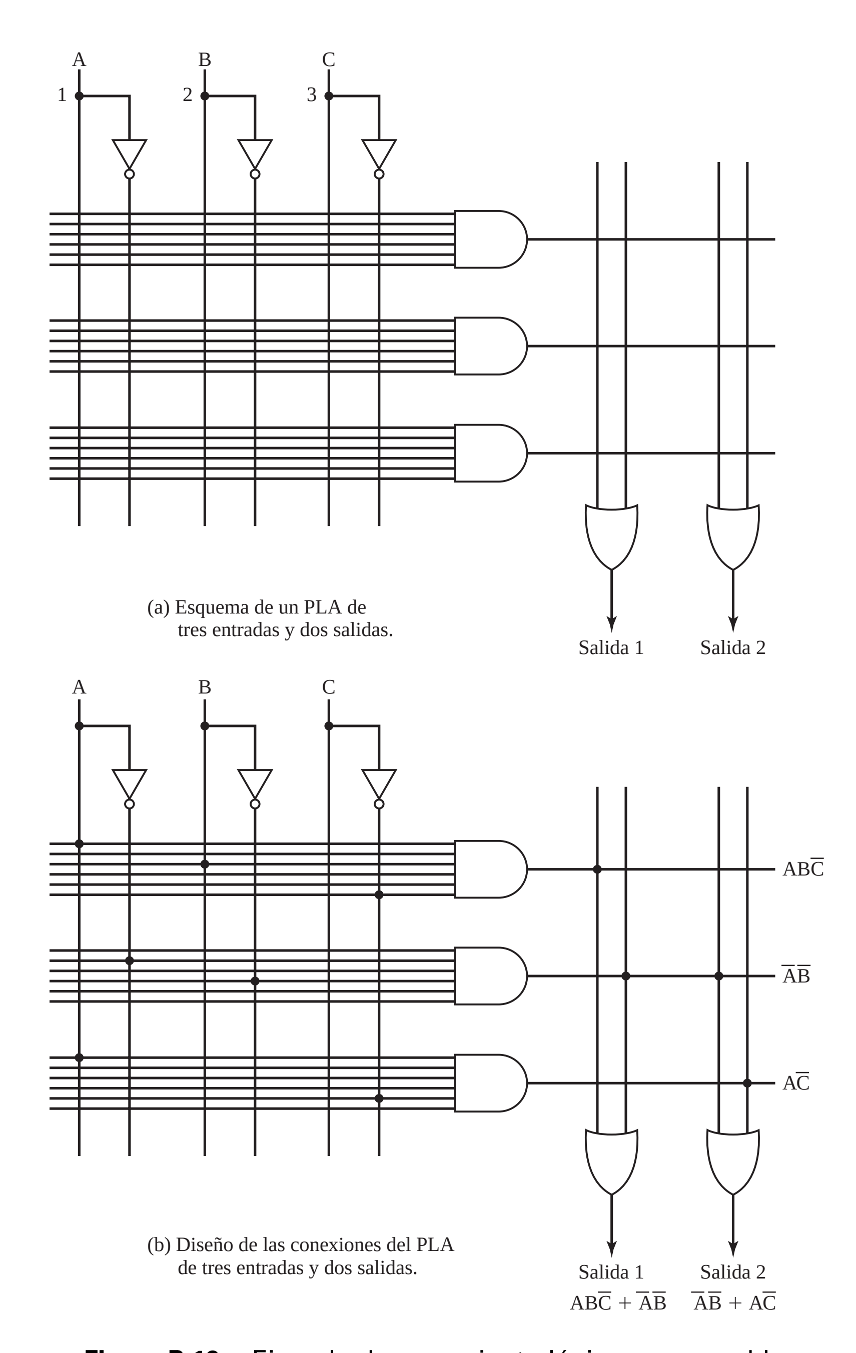


Figura B.19. Ejemplo de un conjunto lógico programable.

Recordemos que una memoria ROM es una unidad de memoria en la que solo se realiza la operación de lectura. Esto implica que la información binaria almacenada en una ROM es permanente y se creó en el proceso de fabricación. Entonces, una entrada dada a la ROM (líneas de direcciones) siempre produce la misma salida (líneas de datos). Como las salidas son función solo de las entradas presentes, la ROM es de hecho un circuito combinacional.

Tabla B.8. Tabla verdad de una ROM.

Entrada				Salida			
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	1	0	0	1
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Se puede implementar una ROM con un decodificador y un conjunto de puertas OR. Como ejemplo, consideremos la Tabla B.8. Esta se puede ver como una tabla verdad con cuatro entradas y cuatro salidas. Para cada uno de los 16 posibles valores de entrada, se muestra el conjunto correspondiente de valores de salida. También se puede ver como la definición del contenido de una ROM de 64 bits de 16 palabras de cuatro bits cada una. Las cuatro entradas determinan una dirección, y las cuatro salidas especifican el contenido de posición indicada en la dirección. En la Figura B.20 se muestra cómo podría implementarse esta memoria usando un decodificador de 4 a 16 y cuatro puertas OR. Como en un PLA, se usa una organización regular y las interconexiones se hacen de forma que reflejen el resultado deseado.

SUMADORES

Hasta ahora hemos visto cómo se pueden usar puertas interconectadas para implementar funciones como enrutamiento de señales, decodificación y ROM. Un área esencial a la que no nos hemos dirigido todavía es la aritmética. En esta breve visión general, nos contentaremos con ver la función de suma.

La suma binaria difiere de la del álgebra booleana en que el resultado incluye un término de acarreo. Así,

$$\begin{array}{r} 0 \\ +0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ +1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +1 \\ \hline 10 \end{array}$$

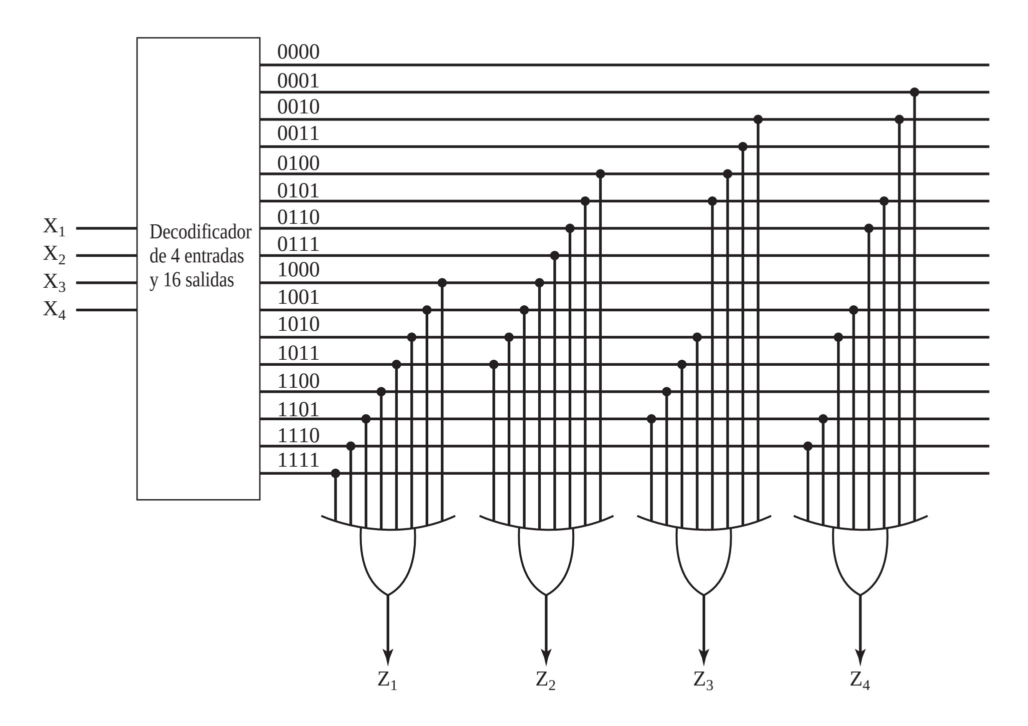


Figura B.20. ROM de 64 bits.

No obstante, la suma se puede implementar con términos booleanos. En la Tabla B.9 puede verse la lógica para sumar dos bits de entrada para producir un bit de suma y un bit de acarreo. Esta tabla verdad podría implementarse fácilmente en lógica digital. Sin embargo, no estamos interesados en realizar la suma con solo un par de bits. Más bien queremos sumar dos números de *n* bits. Esto se puede hacer poniendo juntos un conjunto de sumadores de forma que el acarreo de un sumador sea la entrada del siguiente. En la Figura B.21 se muestra un sumador de cuatro bits.

Tabla B.9. Tabla verdad de la suma binaria.

(a) Suma con un bit			
A	**B**	**Suma**	**Acarreo**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b) Suma con acarreo de entrada				
C_{in}	**A**	**B**	**Sum**	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

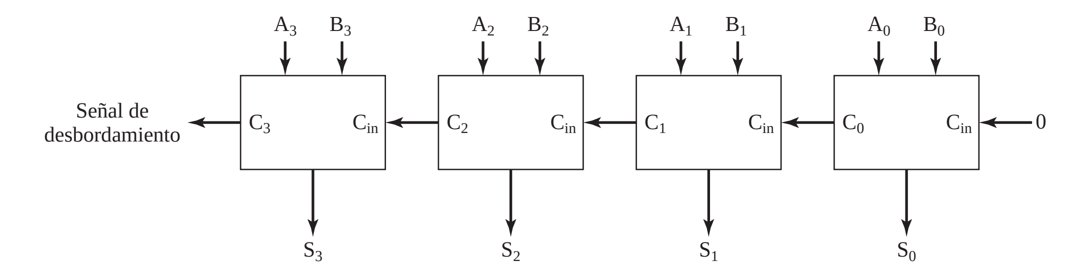


Figura B.21. Sumador de cuatro bits.

Para que funcione un sumador de varios bits, cada uno de los sumadores de un bit debe tener tres entradas, incluyendo el acarreo del sumador inmediato inferior. La tabla verdad revisada aparece en la Tabla B.9b. Las dos salidas se pueden expresar:

$$\text{Suma} = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + ABC + A\overline{B}\overline{C}$$
$$\text{Acarreo} = AB + AC + BC$$

La Figura B.22 es una implementación usando puertas AND, OR y NOT.

Entonces tenemos la lógica necesaria para implementar un sumador de varios bits como se muestra en la Figura B.23. Hay que notar que como la salida de cada sumador depende del acarreo del sumador previo, hay un retardo que crece del bit menos significativo al más significativo. Cada

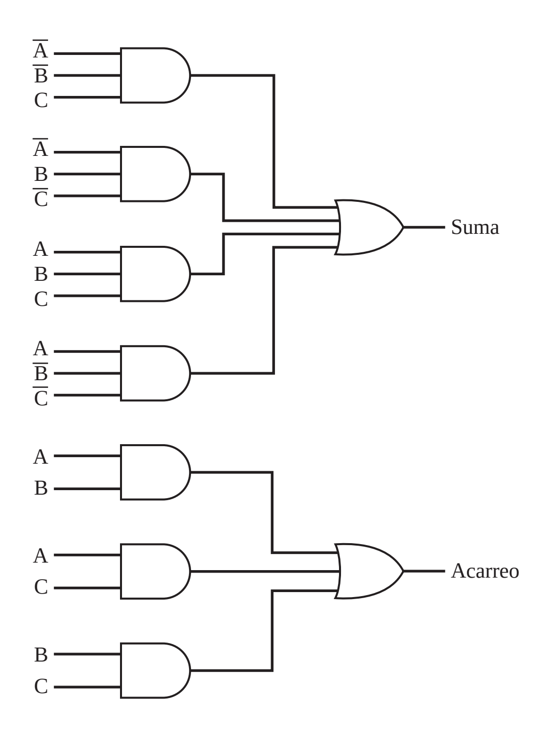


Figura B.22. Implementación de un sumador.

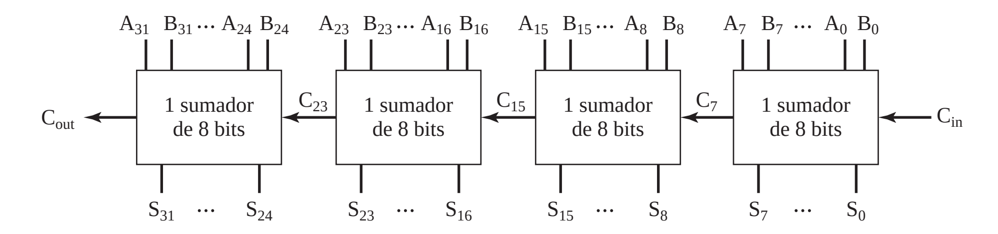


Figura B.23. Construcción de un sumador de 32 bits usando sumadores de 8 bits.

sumador de un bit experimenta cierta cantidad del retardo de puerta, y este retardo de puerta se acumula. Para sumadores grandes, el retardo acumulado puede hacerse inaceptablemente alto.

Si los valores de acarreo se pudieran determinar sin tener que pasar a través de todas las etapas previas, entonces cada sumador de un bit podría funcionar independientemente, y el retardo no se acumularía. Este se puede lograr con una procedimiento conocido como *acarreo anticipado.* Veamos de nuevo el sumador de cuatro bits para explicar este método.

Nos gustaría proponer una expresión que especificara el acarreo de entrada a cualquier etapa del sumador sin referirnos a los valores de acarreo previos. Tenemos

$$C0 = A_0B_0 \quad \text{(B.4)}$$

$$C1 = A_1B_1 + A_1A_0B_0 + B_1A_0B_0 \quad \text{(B.5)}$$

Siguiendo el mismo procedimiento, tenemos

$$C2 = A_2B_2 + A_2A_1B_1 + A_2A_1A_0B_0 + A_2B_1A_0B_0 + B_2A_1B_1 + B_2A_1A_0B_0 + B_2B_1A_0B_0$$

Este proceso se puede repetir para sumadores arbitrariamente grandes. Cada término de acarreo se puede expresar en forma SOP como función de solo las entradas originales, sin dependencia de los acarreos. Por tanto, solo se dan dos niveles de retardo de puerta a pesar del tamaño del sumador.

Para números grandes, este procedimiento se vuelve demasiado complicada. Evaluando la expresión para el bit más significante de un sumador de n bits, se requiere una puerta OR con $n - 1$ entradas y n puertas AND de 2 a $n + 1$ entradas. Por consiguiente, el acarreo anticipado se hace normalmente con solo cuatro u ocho bits a la vez. La Figura B.23 muestra como se puede construir un sumador de 32 bits a partir de cuatro sumadores de ocho bits. En este caso, el acarreo debe pasar a través de los cuatro sumadores de ocho bits, pero puede ser sustancialmente más rápido que pasar a través de 32 sumadores de un bit.

B.4. CIRCUITOS SECUENCIALES

Los circuitos combinacionales implementan las funciones esenciales de un computador digital. Sin embargo, excepto para el caso especial de ROM, no proporcionan memoria o información de estado, elementos también esenciales para el funcionamiento de un computador digital. Para estos últimos fines, se usa una forma de circuitos lógicos digitales más compleja: los circuitos secuenciales. La

salida actual de un circuito secuencial depende no solo de la entrada actual, sino también de la historia pasada de las entradas. Otra manera más general y útil de ver esto es que la salida actual de un circuito secuencial depende de la entrada actual y del estado actual del circuito.

En esta sección, examinaremos algunos ejemplos sencillos pero útiles de circuitos secuenciales. Como veremos, los circuitos secuenciales se implementan con circuitos combinacionales.

BIESTABLES

La forma más sencilla de un circuito secuencial es un biestable. Hay varios tipos de biestables, y todos ellos comparten dos propiedades:

- El biestable es un dispositivo con dos estados. Está en uno de dos estados, en ausencia de entrada, recordando el último estado. Entonces, el biestable puede funcionar como una memoria de un bit.
- El biestable tiene dos salidas, que son siempre complementarias. Normalmente se denominan Q y $\overline{Q}$.

El cerrojo (latch) S-R. La Figura B.24 muestra una configuración común conocida como biestable S-R ó *cerrojo* S-R. El circuito tiene dos entradas, S (Set) y R (Reset), y dos salidas, Q y $\overline{Q}$, y consiste en dos puertas NOR conectadas por realimentación.

Primero, mostremos que el circuito biestable. Supongamos que S y R valen 0 y que Q es 0. Las entradas a la puerta NOR inferior son Q = 0 y S = 0. Entonces, la salida $\overline{Q}$ = 1 significa que las entradas de la puerta NOR superior son $\overline{Q}$ = 1 y R = 0, con salida Q = 0. Por tanto, el estado del circuito es internamente consistente y permanece estable mientras S = R = 0. Con un razonamiento similar se llega a que el estado Q = 1, $\overline{Q}$ = 0, es también estable para R = S = 0.

Por tanto, estos circuitos pueden funcionar como una memoria de 1bit. Podemos ver la salida Q como el «valor» del bit. Las entradas S y R sirven para escribir los valores 1 y 0, respectivamente, en la memoria. Para ver esto, consideremos el estado Q = 0, $\overline{Q}$ = 1, S = 0, R = 0. Supongamos que S cambia al valor 1. Ahora las entradas el la puerta NOR inferior son S = 1, Q = 0. Después de cierto tiempo de retardo (*t*, la salida de la puerta NOR inferior será $\overline{Q}$ = 0 (*ver* Figura B.25).

Así que, en este momento, las entradas a la puerta NOR superior pasan a ser R = 0, $\overline{Q}$ = 0. Después de otro retarde de puerta de (*t*, la salida Q pasa a 1. Este de nuevo un estado estable. Las entradas de la puerta inferior son ahora S = 1, Q = 1, que mantienen la salida Q = 0. Mientras

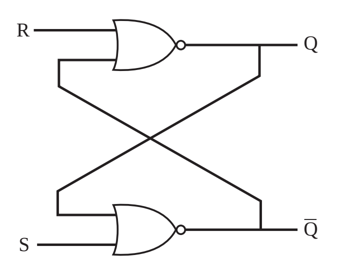


Figura B.24. Implementación del bienestar S-R con puertas NOR.

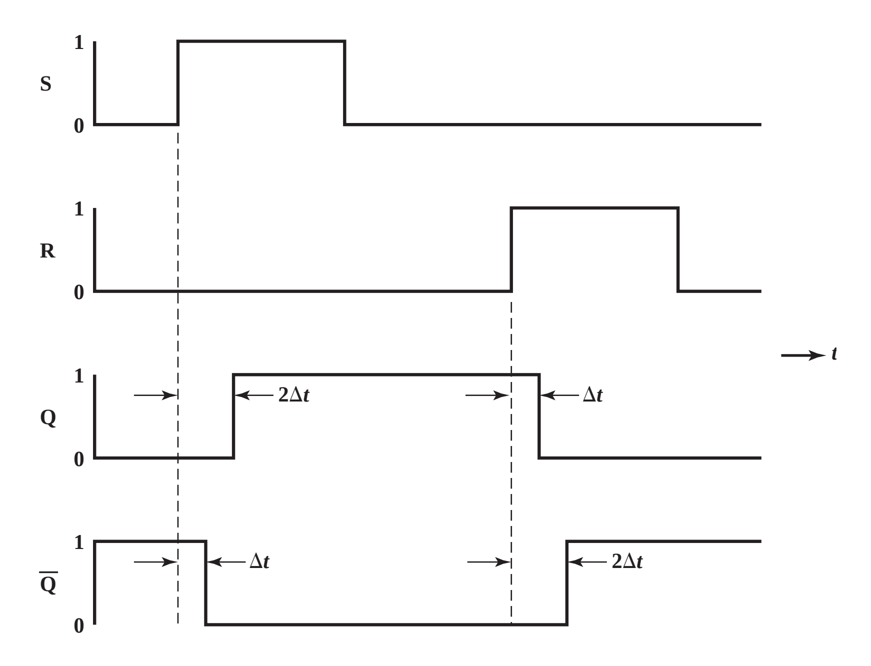


Figura B.25. Diagrama de tiempo del bienestar S-R con NOR.

S = 1 y R = 0, las salidas seguirán siendo Q = 1, $\overline{Q}$ = 0. Además, si S vuelve a 0, las salidas permanecerán sin cambiar.

La salida R realiza la función contraria. Cuando R vale 1, fuerza Q = 0, $\overline{Q}$ = 1, sin importar el estado previo de Q y $\overline{Q}$. De nuevo, hay un tiempo de retardo de $2\Delta t$ antes de que se restablezca la estabilidad (Figura B.25).

El biestable S-R se puede definir con una tabla parecida a la tabla verdad, llamada *tabla característica,* que muestra el siguiente estado o estados de un circuito secuencial en función de los estados y entradas actuales. En el caso del biestable S-R, el estado se puede definir por el valor de Q. La Tabla B.10a muestra la tabla característica resultante. Se observa que las entradas S = 1, R = 1 no están permitidas, ya que producirían una salida inconsistente (Q y $\overline{Q}$ igual a 0). La tabla se puede expresar de forma más compacta, como en la Tabla B.10b. En la Tabla B.10c se muestra una ilustración del comportamiento del biestable S-R.

Biestable S-R síncrono. La salida del cerrojo S-R cambia, tras un breve tiempo de retardo, en respuesta a un cambio en la entrada. Esto se denomina *operación asíncrona*. La mayoría de los acontecimientos en computadores digitales están sincronizados por un pulso de reloj, así que los cambios ocurren solo en un pulso de reloj. La Figura B.26 muestra esta disposición. Este dispositivo se denomina *biestable S-R síncrono.* Nótese que las entradas S y R se aplican a las entradas de las puertas NOR sólo durante el pulso de reloj.

Biestable D. Un problema con los biestables S-R es que la condición R = 1, S = 1 debe ser evitada. Una manera de hacerlo es permitir solo una única entrada. El biestable D lo cumple. La Figura B.27 muestra una implementación con puertas y la tabla característica del biestable D. Usando un inversor, las entradas de no reloj de las dos puertas AND garantizan ser una la opuesta de la otra.

Tabla B.10. El cerrojo S-R.

(a) Tabla característica		
Current Imputs	**Current State**	**Next State**
SR	Q_n	Q_{n+1}
00	0	0
00	1	1
01	0	0
01	1	0
10	0	1
10	1	1
11	0	—
11	1	—

(b) Tabla característica simplificada		
S	**R**	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	—

(c) Respuesta a una serie de entradas										
t	0	1	2	3	4	5	6	7	8	9
S	1	0	0	0	0	0	0	0	1	0
R	0	0	0	1	0	0	1	0	0	0
Q_{n+1}	1	1	1	0	0	0	0	0	1	1

El biestable D a veces se denomina biestable de datos porque es, en efecto, almacén para un bit de datos. La salida del biestable D es siempre igual al valor más reciente aplicado a la entrada. Por tanto, recuerda y produce la última entrada. También se le llama biestable de retardo, porque retrasa un cerp o uno aplicado a la entrada durante un pulso de reloj.

Biestable J-K. Otro biestable útil es el biestable J-K. Como el biestable S-R, tiene dos entradas. Sin embargo, en este caso, todas las combinaciones posibles de los valores de entrada son válidos. La Figura B.28 muestra una implementación con puertas del biestable J-K, y la Figura B.29 muestra su tabla característica (junto con las de los biestables S-R y D). Nótese que las tres primeras

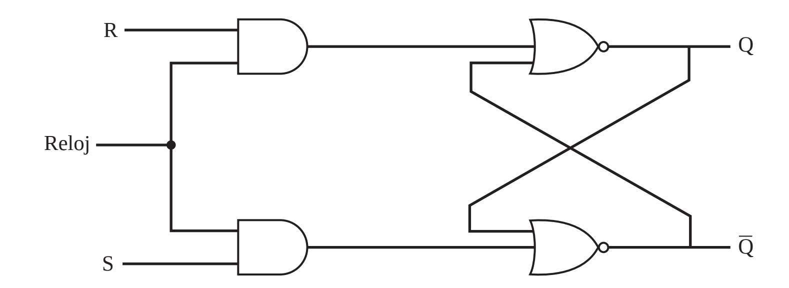


Figura B.26. Bienestable S-R sincronizador.

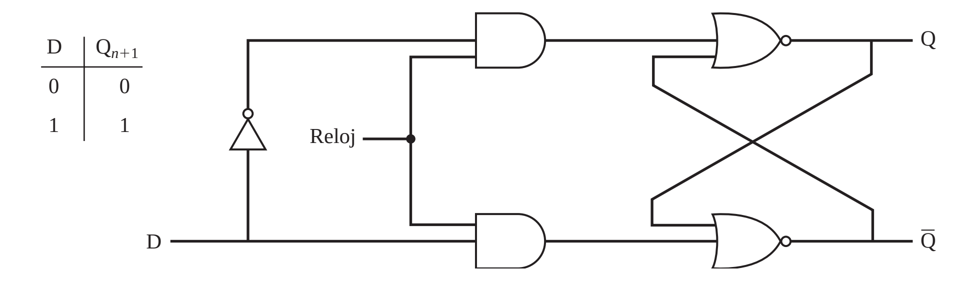


Figura B.27. Biestable D.

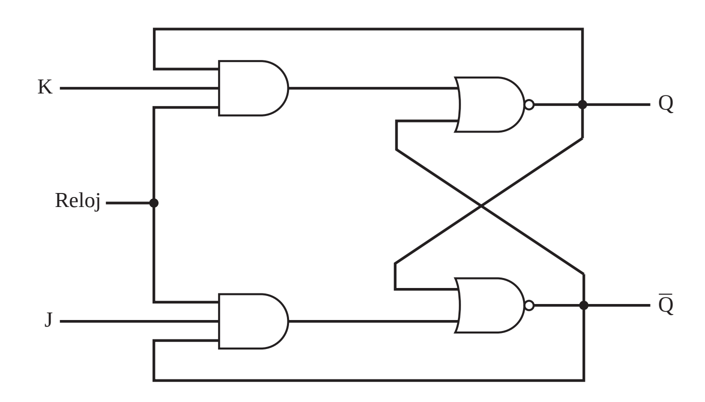


Figura B.28. Biestable J-K.

combinaciones son las mismas que para el biestable S-R. Sin entrada, la salida es estable. La entrada J sola realiza la función de puesta a 1, causando que la salida sea 1; la entrada K solo realiza la función de puesta a cero, provocando que la salida sea 0. Cuando J y K son 1, la función realizada se denomina función de *conmutación*: la salida se invierte.

Si Q vale 1 y se aplica un 1 a J y K, entonces Q se hace 1. El lector debería verificar que la implementación de la Figura B.28 produce esta función característica.

REGISTROS

Como ejemplo del uso de los biestables, examinemos primero uno de los elementos esenciales de la CPU: el registro. Como sabemos, un registro es un circuito digital usado en la CPU para almacenar uno o más bits de datos. Normalmente se usan dos tipos básicos de registros: registros paralelos y de desplazamiento.

Registros paralelos. Un registro paralelo consiste en un conjunto de memorias de un bit que se pueden leer o escribir simultáneamente. Se usa para almacenar datos. Los registros de los que hemos hablado a lo largo de este libro son registros paralelos.

El registro de ocho bits de la Figura B.30 ilustra el funcionamiento de un registro paralelo usando biestables D. Una señal de control, llamada *validación de dato de entrada*, controla la escritura en

Nombre	Símbolo gráfico	Tabla característica
S–R	S, Q, Ck, R, $\overline{Q}$	S R Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 –
J–K	J, Q, Ck, K, $\overline{Q}$	J K Q_{n+1} 0 0 Q_n 0 1 0 1 0 1 1 1 $\overline{Q_n}$
D	D, Q, Ck, $\overline{Q}$	D Q_{n+1} 0 0 1 1

Figura B.29. Biestables básicos.

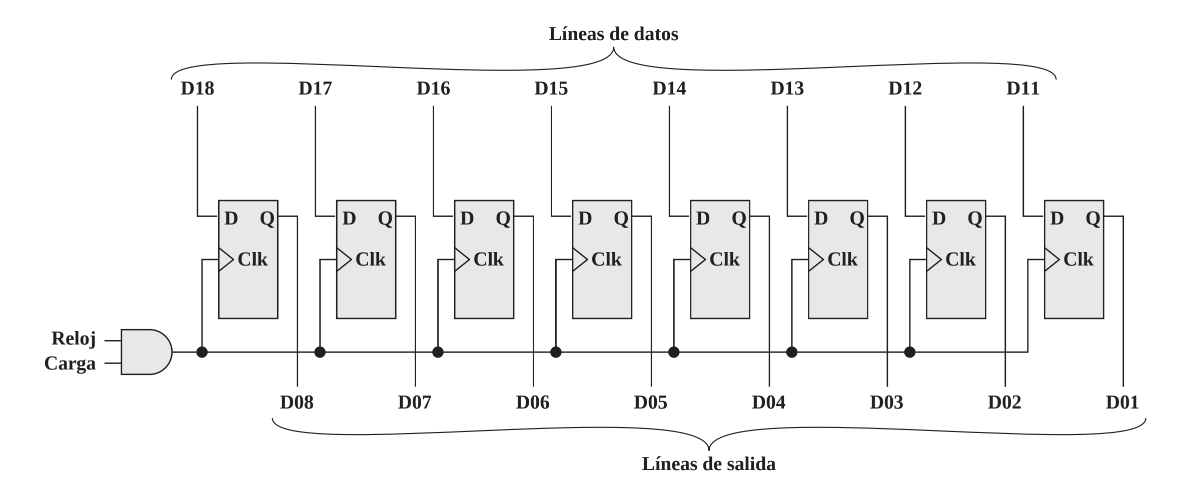


Figura B.30. Registro paralelo de ocho bits.

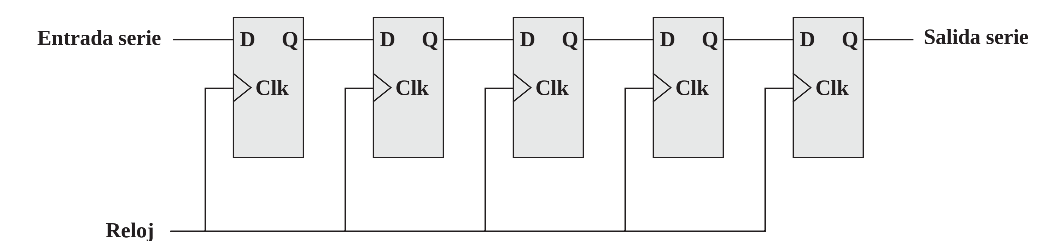


Figura B.31. Registro de desplazamiento de cinco bits.

los registros de los valores provenientes de las líneas de señales, de la D11 a la D18. Estas líneas pueden ser salidas de multiplexores, ya que los datos que se pueden cargar en un registro pueden provenir de una gran variedad de fuentes.

Registro de desplazamiento. Un registro de desplazamiento acepta y/o transfiere información vía serie. Consideremos, por ejemplo, la Figura B.31, que muestra un registro de desplazamiento de cinco bits construido a partir de biestables D síncronos. Los datos se introducen únicamente a través del biestable que está más a la izquierda. Con cada pulso de reloj, los datos se desplazan a la derecha una posición, y el bit más a la derecha se transfiere fuera.

Los registros de desplazamiento se pueden usar como interfaz de dispositivos serie de E/S. Además, pueden usarse en la ALU para realizar desplazamiento lógicos y funciones de rotación. En este último uso, necesitan equiparse con circuitería de lectura/escritura tanto paralela como serie.

CONTADORES

Otra categoría útil de circuitos secuenciales es la de contador. Un contador es un registro cuyo valor se puede incrementar fácilmente en 1, módulo la capacidad del registro. Así, un registro hecho con n biestables puede contar hasta $2^n - 1$. Cuando el contador se incrementa más allá de su valor máximo, se pone a 0. Un ejemplo de un contador en la CPU es el contador de programa.

Los contadores pueden ser asíncronos o síncronos, dependiendo de la forma en que operen. Los contadores asíncronos son relativamente lentos, ya que la salida de un biestable produce un cambio en el estado del siguiente biestable. En un contador síncrono, todos los biestables cambian de estado a la vez. Como el último tipo es mucho más rápido, es el tipo que se usa en CPU. Sin embargo, es útil empezar la discusión con una descripción del contador asíncrono.

Contador asíncrono. Un contador asíncrono se denomina también contador de onda (*ripple*), ya que el cambio que se produce para incrementar el contador empieza en un extremo y se transfiere como una «onda» hasta el otro extremo. La Figura B.32 muestra una implementación de un contador de cuatro bits usando biestables J-K, junto con un diagrama de tiempo que ilustra su comportamiento. El diagrama de tiempo está idealizado ya que no muestra el retardo de propagación que se produce cuando las señales bajan en la serie de biestables. La salida del biestable más a la izquierda (Q_0) es el bit menos significante. El diseño se podría ampliar fácilmente a un número arbitrario de bits añadiendo en cascada más biestables.

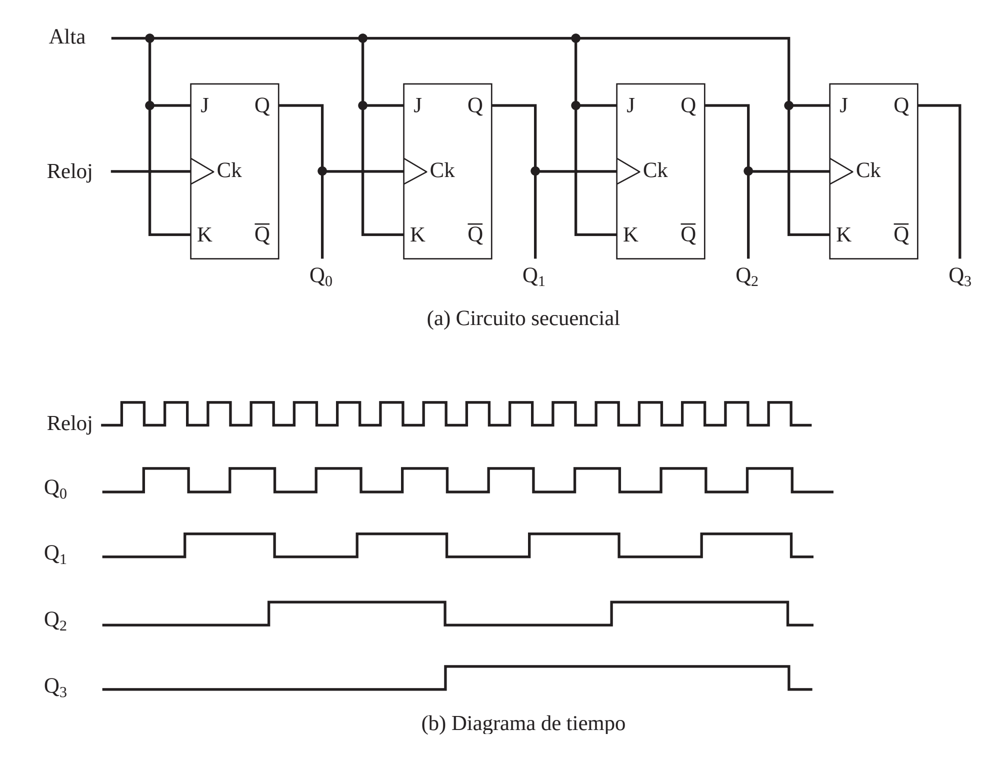
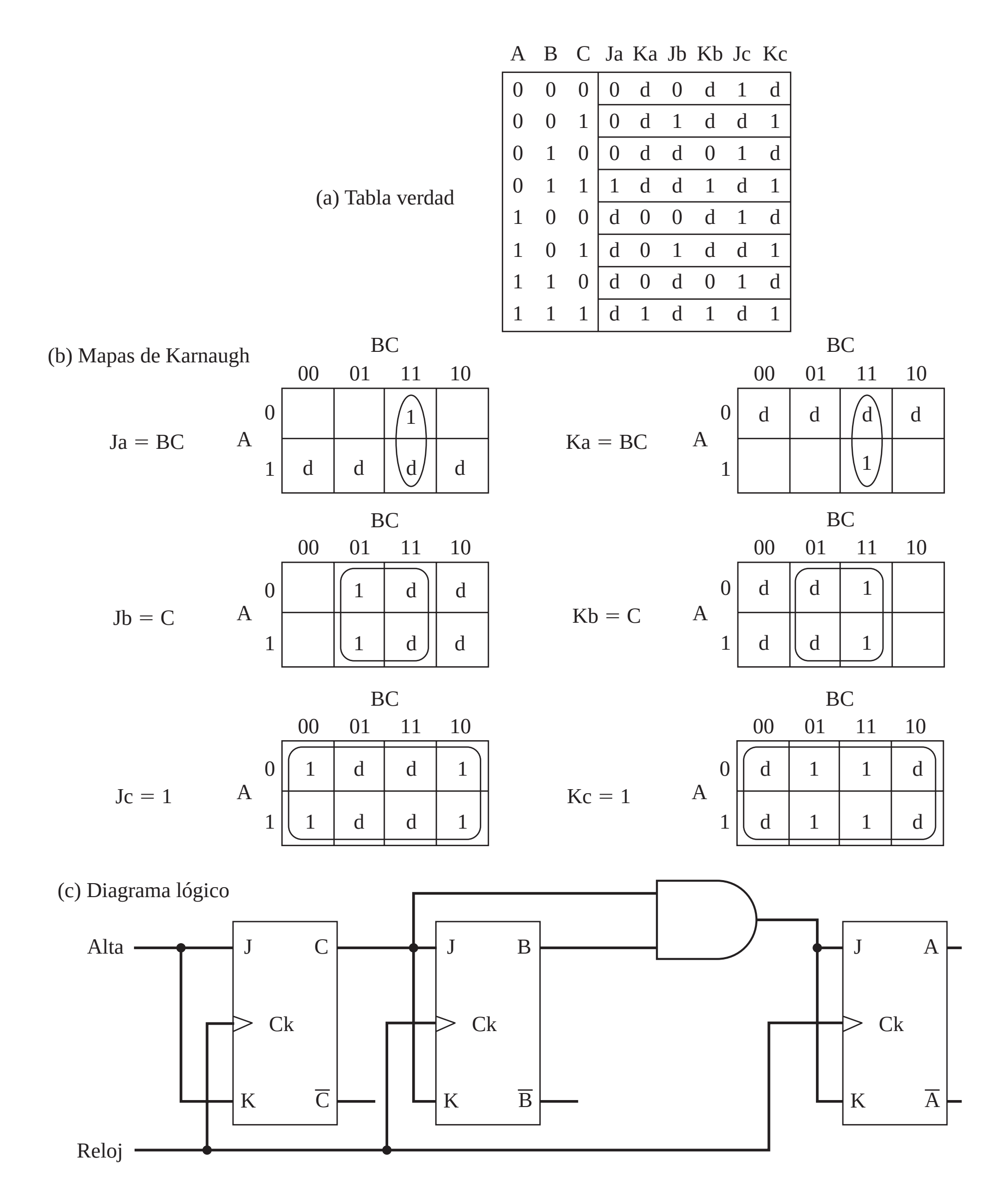


Figura B.32. Contador ondulado.

En la implementación mostrada, el contador se incrementa con cada pulso de reloj. Las entradas J y K de cada biestables se mantienen a 1 constante. Esto quiere decir que, cuando hay un pulso de reloj, la salida en Q se invierte (de 1 a 0; de 0 a 1). Nótese que el cambio de estado se produce cuando cae el flanco del pulso de reloj; esto se conoce como biestable disparado por flanco. Usando biestables que responden a la transición en un pulso de reloj, en vez de en el pulso mismo, proporciona un control del tiempo mejor en circuitos complejos. Si se analizan los patrones de salida de este contador, se puede ver el ciclo 0000, 0001..., 1110, 1111, 0000, etc.

Contadores síncronos. El contador asíncrono tiene la desventaja del retardo asociado al cambio de valor, que es proporcional al tamaño del contador. Para superar esta desventaja, las CPU usan contadores síncronos, en los que todos los biestables del contador cambian al mismo tiempo. En esta subsección, presentamos el diseño de un contador síncrono de tres bits. Al mismo tiempo, ilustraremos algunos conceptos básicos en el diseño de circuitos síncronos.

Para un contador de tres bits, necesitamos tres biestables. Vamos a usar biestables J-K. Llamemos a las salidas sin complementar de los tres biestables A, B, y C respectivamente, donde C representa el bit menos significativo. El primer paso es construir la tabla verdad que relaciona las entradas J-K con las salidas, para poder diseñar el resto del circuito. Dicha tabla verdad se muestra en la Figura B.33a. Las primeras tres columnas muestran las combinaciones posibles de las salidas A, B y C. Están listadas en el orden en que aparecen cuando se incrementa el contador. Cada fila indica el valor actual de A, B, C y las entradas a los tres biestables que se necesitarán para obtener el próximo valor de A, B, y C.



(a) Tabla verdad

A	B	C	Ja	Ka	Jb	Kb	Jc	Kc
0	0	0	0	d	0	d	1	d
0	0	1	0	d	1	d	d	1
0	1	0	0	d	d	0	1	d
0	1	1	1	d	d	1	d	1
1	0	0	d	0	0	d	1	d
1	0	1	d	0	1	d	d	1
1	1	0	d	0	d	0	1	d
1	1	1	d	1	d	1	d	1

Figura B.33. Diseño de un contador síncrono.

Para comprender la forma en la que se ha hecho la tabla verdad de la Figura B.33a, puede ser útil reestructurar la tabla característica del biestable J-K. Recordemos que esta tabla verdad se presentaba como sigue:

J	**K**	$\mathbf{Q_{n+1}}$
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_{n+1}$

En esta forma, la tabla muestra el efecto que las entradas J y K tienen en la salida. Ahora consideremos la siguiente reorganización de la misma información:

Q_n	J	K	Q_{n+1}
0	0	d	0
0	1	d	1
1	d	1	0
1	d	0	1

En esta forma, la tabla proporciona el valor de la siguiente salida cuando se conocen las entradas y la salida actual. Esta es exactamente la información que se necesita para diseñar el contador o, de hecho, cualquier circuito secuencial. En esta forma, la tabla se denomina *tabla de excitaciones.*

Volvamos a la Figura B.33a. Consideremos la primera fila. Queremos que el valor de A permanezca a 0, que el valor de B a 0, y que el valor de C cambie de 0 a 1 con la siguiente aplicación del pulso de reloj. La tabla de excitaciones muestra que para mantener una salida a 0, debemos tener las entradas J = 0 e indiferencia para K. Para que haya una transición de 0 a 1, las entradas deben ser J = 1 y K = d. Estos valores se encuentran en la primera fila de la tabla. Con un razonamiento similar, se puede rellenar el resto de la tabla.

Una vez construida la tabla verdad de la Figura B.33a, vemos que la tabla muestra los valores requeridos para todas las entradas J y K como funciones de los valores actuales de A, B, y C. Con la ayuda de mapas de Karnaugh, podemos obtener las expresiones booleanas para estas seis funciones. Esto se muestra en la parte b de la figura. Por ejemplo, el mapa de Karnaugh de la variable Ja (la entrada J del biestable que produce la salida A) da lugar a la expresión Ja = BC. Cuando se derivan las seis expresiones, es un problema sencillo diseñar el circuito real, como se muestra en la parte c de la figura.

B.5. LECTURAS RECOMENDADAS Y SITIOS WEB

[GREG98] es una introducción fácil de leer sobre los conceptos de este capítulo. [STON96] es una excelente introducción corta. Hay una serie de libros de texto que hacen un tratamiento más profundo; como [MANO04] y [FARH04].

FARH04 Farhat, H. *Digital Design and Computer Organization.* Boca Raton: CRC Press, 2004.

GREG98 Gregg, J. *Ones and Zeros: Understanding Boolean Algebra, Digital Circuits, and the Logic of Sets.* New York: Wiley, 1998.

MANO04 Mano, M., and Kime, C. *Logic and Computer Design Fundamentals.* Upper Saddle River, NJ: Prentice Hall, 2004.

STON96 Stonham, T. *Digital Logic Techniques.* London: Chapman & Hall, 1996.

SITIO WEB RECOMENDADO

- **Lógica digital:** colección útil de diagramas de circuitos interactivos.

B.6. PROBLEMAS

B.1. Construir la tabla verdad de las siguientes expresiones booleanas:

(a) $ABC + \overline{A}\,\overline{B}\,\overline{C}$
(b) $ABC + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C}$
(c) $A(\overline{B}C + \overline{B}C)$
(d) $(A + B)(A + C)(\overline{A} + \overline{B})$

B.2. Simplificar las siguientes expresiones aplicando la ley conmutativa:

(a) $A \cdot \overline{B} + \overline{B} \cdot A + C \cdot D \cdot E + \overline{C} \cdot D \cdot E + E \cdot \overline{C} \cdot D$
(b) $A \cdot B + A \cdot C + B \cdot A$
(c) $(L \cdot M \cdot N)(A \cdot B)(C \cdot D \cdot E)(M \cdot N \cdot L)$
(d) $F \cdot (K + R) + S \cdot V + W \cdot \overline{X} + V \cdot S + \overline{X} \cdot W + (R + K) \cdot F$

B.3. Aplicar el teorema de DeMorgan a las siguientes ecuaciones:

(a) $F = \overline{V + A + L}$
(b) $F = \overline{A} + \overline{B} + \overline{C} + \overline{D}$

B.4. Simplificar las siguientes expresiones:

(a) $A = S \cdot T + V \cdot W + R \cdot S \cdot T$
(b) $A = T \cdot U \cdot V + X \cdot Y + Y$
(c) $A = F \cdot (E + F + G)$
(d) $A = (P \cdot Q + R + S \cdot T)T \cdot S$
(e) $A = \overline{\overline{D} \cdot \overline{D} \cdot E}$
(f) $A = Y \cdot (W + X + \overline{\overline{Y} + \overline{Z}}) \cdot Z$
(g) $A = (B \cdot E + C + F) \cdot C$

B.5. Obtener la operación XOR a partir de las operaciones booleanas básicas AND, NOR y NOT.

B.6. Dibujar el diagrama lógico de una función AND de tres entradas con puertas NOR y NOT.

B.7. Escribir la expresión Booleana de una puerta NAND de cuatro entradas.

B.8. Se usa un circuito combinacional para controlar un visualizador de dígitos decimales de 7 segmentos, como se muestra en la Figura B.34. El circuito tiene cuatro entradas, que proporciona el código de 4 bits usado en representación decimal compacta (010 = 0000, ..., 910 = 1001). Las siete salidas definen que

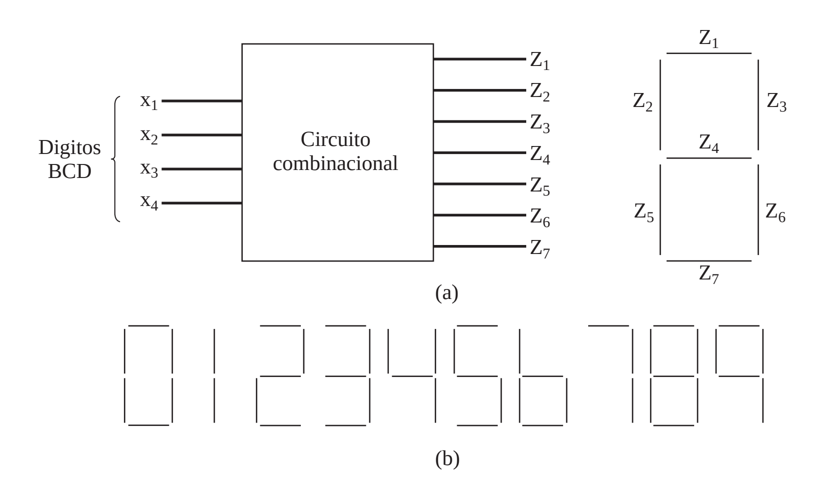


Figura B.34. Ejemplo de un visualizador LED de siete segmentos.

segmentos se van a activar para visualizar un dígito decimal dado. Nótese que no se necesitan algunas combinaciones de entrada ni sus salidas.

(a) Desarrollar la tabla verdad para este circuito.
(b) Expresar la tabla verdad en la forma SOP.
(c) Expresar la tabla verdad en la forma POS.

B.9. Diseñar un multiplexor de 8 a 1.

B.10. Añadir una línea adicional a la Figura B.15 para que funcione como un demultiplexor.

B.11. El código Gray es un código binario para enteros. Difiere de la representación binaria ordinaria en que solo cambia un único bit entre la representación de dos números cualesquiera. Esto es útil en aplicaciones como contadores o conversores analógico digitales donde se genera una secuencia de números. Como solo cambia un bit

Código binario	Código gray
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Diseñar un circuito que convierta un código binario en un código Gray.

B.12. Diseñar un decodificador de 5×32 usando cuatro decodificadores de 3×8 (con entradas de habilitación) y un decodificador de 2×4.

B.13. Implementar el sumador completo de la Figura B.22 con solo cinco puertas (*Ayuda:* algunas de las puertas son XOR).

B.14. Considerar la Figura B.22. Suponer que cada puerta produce un retardo de 10ns. Entonces, la salida de suma es válida tras 30 ns y la salida de acarreo tras 0 ns. ¿Cuál es el tiempo total de suma de un sumador de 32 bits

(a) implementado sin acarreo anticipado, como en la Figura B.21?
(b) implementado con acarreo anticipado y usando sumadores de 8 bits, como en la Figura B.23?

APÉNDICE C

Proyectos para enseñar arquitectura y organización de computadores

Muchos profesores creen que la investigación o la implementación de proyectos son cruciales para entender claramente los conceptos de la arquitectura y organización de computadores. Sin proyectos, puede ser difícil para los estudiantes comprender algunos conceptos e interacciones básicos entre componentes. Los proyectos refuerzan los conceptos introducidos en el libro, dan a los estudiantes una mejor apreciación del funcionamiento interno del procesador, y puede motivar a los estudiantes y darles confianza en que dominan el material.

En este texto se ha tratado de presentar los conceptos lo más claramente posible y de proporcionar más de doscientos ejercicios para reforzar estos conceptos. Muchos profesores desearían complementar este material con proyectos. Este apéndice constituye una guía en este sentido y describe el material de ayuda disponible en el manual del profesor. El material de ayuda cubre tres tipos de proyectos:

- Proyectos de investigación
- Proyectos de simulación
- Lecturas/trabajos

C.1. PROYECTOS DE INVESTIGACIÓN

Una forma efectiva de reforzar los conceptos básicos del curso y enseñar a los estudiantes habilidades para investigar, es asignar proyectos de investigación. Un proyecto de este tipo puede implicar búsqueda de literatura así como productos en venta en la web, tareas de investigación en laboratorio, y esfuerzos de normalización. Los proyectos podrían asignarse a grupos de alumnos, los pequeños proyectos a individuos. En cualquier caso, es mejor exigir algún tipo de propuesta de proyecto al principio, dando tiempo al profesor para evaluarla, ajustar el tema y el nivel de esfuerzo. El anuncio a los estudiantes de los proyectos de investigación deben incluir:

- Un formato para la propuesta
- Un formato para el informe final
- Un esquema con las fechas tope intermedias y finales
- Una lista de posibles temas de proyectos

Los estudiantes pueden seleccionar uno de los temas de la lista o idear su propio proyecto. El manual del profesor incluye un posible formato para la propuesta y el informe final, así como una lista de posibles temas de investigación.

C.2. PROYECTOS DE SIMULACIÓN

Un forma excelente de conseguir una comprensión del comportamiento interno de un procesador y de estudiar y apreciar algunos de los compromisos de diseño e implicaciones de las prestaciones es simulando los elementos clave del procesador. Dos herramientas muy útiles para este propósito son SimpleScalar y SMPCache.

Comparado con la implementación real del hardware, la simulación proporciona ventajas tanto en investigación como en educación:

- Con la simulación, es fácil modificar varios elementos de una organización, variar las características de funcionamiento de varios componentes y luego analizar los efectos de dicha modificación.
- La simulación posibilita la obtención de una serie de estadísticas de funcionamiento detallado, que se puede usar para comprender los compromisos de funcionamiento.

SIMPLESCALAR

SimpleScalar [BURG97, MANJ01a, MANJ01b] es un conjunto de herramientas para simular programas reales dentro de un gran rango de procesadores y sistemas modernos. La herramienta incluye un compilador, un ensamblador, un enlazador y otras herramientas de simulación y visualización. SimpleScalar suministra simuladores de procesadores con una gama que va desde un simulador funcional extremadamente rápido, hasta un simulador de procesador superescalar, con emisión fuera de orden, que soporta cachés no bloqueantes y ejecución especulativa. La arquitectura del conjunto de instrucciones y los parámetros de organización se pueden modificar para crear distintos experimentos.

El manual del profesor para este libro incluye una introducción concisa a SimpleScalar para estudiantes, con instrucciones de cómo cargar e iniciar Simple Scalar. El manual también sugiere algunos proyectos.

SimpleScalar es un paquete software intercambiable que vale para la mayoría de las plataformas UNIX. El software SimpleScalar se puede cargar desde el sitio web SimpleScalar. Está disponible gratis para uso no comercial.

SMPCACHE

SMPCache es un simulador dirigido por trazas para el análisis y enseñanza de la memoria caché de sistemas con multiprocesadores simétricos [RODR01]. La simulación se basa en un modelo construido de acuerdo con los principios básicos arquitecturales de estos sistemas. El simulador tiene una interfaz completamente gráfica y fácil de usar. Algunos de los parámetros que se pueden estudiar con este simulador son: situación del programa, influencia del número de procesadores, protocolos de coherencia caché, esquemas de arbitraje de buses, proyecciones, políticas de sustitución, tamaño de la caché (bloques de la caché), número de conjuntos de cachés (para definir cachés asociativas) y número de palabras por bloque (tamaño de un bloque de memoria).

El manual del profesor de este libro incluye una introducción concisa al SMPCache para estudiantes, con instrucciones sobre cómo cargar y arrancar el SMPCache. El manual también incluye algunas sugerencias sobre asignación de proyectos.

SMPCache es un paquete software intercambiable que funciona en sistemas PC con Windows. El software SMPCache se puede descargar del sitio web del SMPCache. Está disponible gratis para uso no comercial.

C.3. ASIGNACIÓN DE LECTURAS/TRABAJOS

Otra forma excelente de reforzar los conceptos del curso y dar a los estudiantes experiencia en investigación es asignarles artículos para leerlos y analizarlos. El sitio web del profesor incluye una lista de artículos recomendados para asignarlos, organizados por capítulos. Todos los artículos están disponibles a través de Internet como en la biblioteca técnica de cualquier facultad buena. El sitio web del profesor también incluye la terminología recomendada.

GLOSARIO

Acceso directo a memoria (DMA) Forma de E/S que utiliza un módulo especial llamado *módulo DMA*, que controla los intercambios de datos entre la memoria principal y un módulo de E/S. La CPU envía una petición de transferencia de un bloque de datos al módulo DMA y es interrumpido sólo después de que el bloque entero haya sido transferido.

Acceso directo Capacidad de extraer o introducir datos de/en un dispositivo de almacenamiento en una secuencia independiente de sus posiciones relativas; es decir, de las direcciones que indican la posición física del dato.

Acumulador Nombre del registro de la CPU en formato de instrucción de dirección única. El acumulador, o AC, es implícitamente uno de los dos operandos de la instrucción.

Arbitraje del bus Proceso para determinar a que controlador del bus intentando acceder se le permitirá el acceso al bus.

Array lógico programable (PLA) Array de puertas cuyas interconexiones pueden programarse para realizar una función lógica específica.

Array redundante de discos independientes (RAID) Discos múltiples en los que parte de la capacidad de almacenamiento se usa para guardar información redundante sobre datos almacenados por el usuario en el resto de la memoria. La información redundante posibilita la regeneración de los datos del usuario en el caso de que falle uno de los discos del array o el camino de acceso a ellos.

ASCII (American Standard Code for Information Interchange) Código Estándar Americano para Intercambio de Información. El ASCII es un código de 7 bits usado para representar caracteres imprimibles numéricos, alfabéticos y especiales. También incluye códigos para *caracteres de control*, que no se pueden imprimir o visualizar pero que especifican alguna función de control.

Autoindexación Una forma de direccionamiento indexado en el que se incrementa o decrementa automáticamente el registro índice con cada referencia a memoria.

Base En el sistema de numeración usado comúnmente en documentos científicos, es el número que se eleva a la potencia indicada por el exponente y se multiplica por la mantisa para determinar el número real representado. (Por ejemplo, el número 6.25 en la expresión $2{,}7 \times 6{,}25^{1.5} = 42{,}1875$).

Biestable (flip-flop) Circuito o dispositivo que contiene elementos activos, capaz de entrar en uno de dos estados estables posibles en un momento dado. Sinónimo de *circuito biestable*, *báscula.*

Bit de paridad Dígito binario añadido a un grupo de dígitos binarios para hacer la suma de todos los dígitos (unos) o bien impares (paridad impar) o bien pares (paridad par).

Bit En el sistema de numeración binario puro, es 0 o 1.

Bloque de control de un proceso Manifestación de un proceso en un sistema operativo. Es una estructura de datos que contiene información sobre las características y estado del proceso.

Buffer Memoria usada para compensar una diferencia de velocidad en el flujo de datos, o de tiempo de aparición de eventos, cuando se transfieren datos de un dispositivo a otro.

Bus Camino de comunicación compartido, consistente en una o varias líneas. En algunos computadores, la CPU, la memoria, y los componentes de E/S se conectan a un bus común. Como las líneas son compartidas por varios componentes, sólo uno puede transmitir a la vez.

Bus de control Parte del bus para transferir señales de control.

Bus de datos Parte de un bus usada para transferir datos.

Bus de direcciones Porción de un bus del sistema usada para transferir una dirección. Típicamente, la dirección identifica una posición de la memoria principal o un dispositivo de E/S.

Bus del sistema Bus que se usa para interconectar varios componentes (CPU, memoria, E/S).

Byte Ocho bits. A veces se denomina *octeto*.

Caché Memoria relativamente pequeña y rápida situada entre una memoria grande y lenta y la lógica que accede a la memoria grande. La caché contiene los datos a los que se ha accedido recientemente, y se diseña para aumentar la velocidad en los accesos a los mismos datos.

Canal de E/S Módulo, relativamente complejo, que releva a la CPU de los detalles de las operaciones de E/S. Un canal de E/S ejecuta una secuencia de órdenes de E/S de la memoria principal sin necesidad de la implicación de la CPU.

Canal multiplexor Canal diseñado para operar con varios dispositivos de E/S simultáneamente. Varios dispositivos de E/S pueden transferir registros al mismo tiempo intercalando campos de datos. Ver también *canal multiplexor de bytes, canal multiplexor de bloques.*

Canal multiplexor de bloques Canal multiplexor que intercala bloques de datos. Ver también *canal multiplexor de bytes*. Contrastar con *canal selector.*

Canal multiplexor de bytes Canal multiplexor que intercala bytes de datos. Ver también *canal multiplexor de bloques*. Contrastar con *canal selector.*

Canal selector Canal de E/S diseñado para operar sólo con un dispositivo de E/S a la vez. Una vez se haya seleccionado el dispositivo de E/S, se transfiere un registro completo byte a byte. Contrastar con *canal multiplexor de bloques, canal multiplexor.*

CD-ROM (Compact Disk Read-Only Memory) Disco compacto de solo lectura usado como memoria de datos. El sistema estándar usa discos de 12 cm que pueden almacenar más de 550 Mbytes.

Ciclo de captación Parte de un ciclo de instrucción en la que la CPU capta de la memoria la instrucción que se va a ejecutar.

Ciclo de ejecución Parte de un ciclo de instrucción en la que la CPU ejecuta la operación especificada por el código de operación de la instrucción.

Ciclo de instrucción Proceso realizado por la CPU para ejecutar una instrucción.

Ciclo de interrupción Parte del ciclo de instrucción durante la que la CPU comprueba las interrupciones. Si hay pendiente una interrupción habilitada, la CPU salva el estado actual del programa y procesa una rutina que gestiona la interrupción.

Ciclo indirecto Parte del ciclo de instrucción durante la que la CPU realiza un acceso a memoria para convertir una dirección indirecta en directa.

Cinta magnética Cinta con una superficie magnetizable donde se pueden almacenar datos grabándolos magnéticamente.

Circuito combinacional Dispositivo lógico cuyos valores de salida, en cualquier instante, dependen sólo de los valores de entrada en ese instante. Un circuito combinacional es un caso especial de circuito secuencial que no tiene capacidad de almacenamiento.

Circuito integrado (IC) Pequeño trozo de material sólido, por ejemplo, silicio, sobre el que se graban o imprimen una serie de componentes electrónicos y sus interconexiones.

Circuito secuencial Circuito lógico cuya salida depende de la entrada actual y del estado del circuito. Los circuitos secuenciales poseen, por tanto, el atributo de memoria.

Cluster Grupo de computadores completos interconectados, trabajando a la vez como un recurso unificado que puede crear la ilusión de ser una única máquina. El término *computador completo* se refiere a un sistema que puede funcionar por sí solo, sin conexión al cluster.

Código de condición Bit que refleja el resultado de una operación (por ejemplo, aritmética). La CPU puede incluir uno o más bits de condición, que se pueden almacenar de forma separada en la CPU o como parte de un registro de control. También se le llama *indicador.*

Código de corrección de errores Código en el que cada carácter o señal se establece conforme a unas reglas específicas de construcción, de manera que las desviaciones de estas reglas indican la presencia de un error, y algunos o todos los errores detectados pueden corregirse automáticamente.

Código de detección de errores Código en el que cada carácter o señal se establece conforme a unas reglas específicas de construcción, de forma que las desviaciones de estas reglas indican la presencia de un error.

Código de operación Código usado para representar las operaciones de un computador. Se denomina abreviadamente codop.

Codop Abreviatura de *código de operación.*

Componente de estado sólido Componente cuyo funcionamiento depende del control de un fenómeno eléctrico o magnético en un sólido (por ejemplo, transistor, diodo de cristal, núcleo de ferrita).

Computación de altas prestaciones (HPC, "*High performance computing*") Área de investigación sobre supercomputadores y su software. Se hace énfasis en las aplicaciones científicas, que pueden implicar el uso de computación vectorial y matricial, y algoritmos paralelos.

Comunicación de datos Transferencia de datos entre dispositivos. El término generalmente excluye E/S.

Contador de programa Registro que contiene la dirección de la instrucción a ejecutar.

Controlador de E/S Módulo de E/S relativamente sencillo, que requiere un control de la CPU o de un canal de E/S. Es sinónimo de *controlador de dispositivo.*

Controlador del bus Dispositivo asociado al bus que es capaz de iniciar y controlar la comunicación en el bus.

CPU microprogramada CPU cuya unidad de control se implementa usando microprogramación.

Decodificador Dispositivo en el que las líneas de entrada representan un código, y sólo se activa una de las líneas de salida, existiendo una correspondencia una a una entre las líneas de salida y los códigos de entrada.

Demanda de página Transferencia de una página desde una memoria auxiliar a una memoria real en el momento en que se necesite.

Dirección absoluta Dirección, en lenguaje de los computadores, que identifica una posición de almacenamiento o un dispositivo sin usar ninguna referencia intermedia.

Dirección base Valor numérico que se usa como referencia en el cálculo de direcciones en la ejecución de un programa.

Dirección directa Dirección que designa la posición de almacenamiento de un dato que va a ser tratado como operando. Sinónimo de *dirección de un nivel.*

Dirección indexada Dirección que es modificada por el contenido de un registro índice antes de, o durante, la ejecución de una instrucción.

Dirección indirecta Dirección de una posición de memoria que contiene a su vez otra dirección.

Dirección inmediata Contenido de una parte de una dirección que contiene el valor de un operando en lugar de una dirección. Es sinónimo de *dirección de nivel cero.*

Disco compacto (CD) Disco no borrable que almacena información de audio digitalizada.

Disco en tiras de datos (Disk stripping) Método de asignación en discos múltiples en el cual se asignan bloques lógicamente contiguos de datos, o tiras, cíclicamente a discos consecutivos. Un conjunto de tiras lógicamente consecutivas que se asignan exactamente a una tira en cada disco se denomina barra.

Disco magnético Sustrato circular (plato) plano con una superficie magnetizable en una o ambas caras donde se pueden almacenar datos.

Disco óptico borrable Disco que usa una tecnología óptica en la que una información puede ser fácilmente borrada y reescrita. Hay dos tipos en uso el de 3.25 y 5.25 pulgadas. Una capacidad típica es 650 Mbytes.

Disquete Disco magnético flexible encerrado en un contenedor protector. Sinónimo de *disco flexible.*

E/S aislada Método de direccionamiento de módulos de E/S y dispositivos externos. El espacio de direcciones de E/S se trata separadamente del espacio de direcciones de la memoria principal. Se usan instrucciones máquina específicas de E/S. Comparar con *E/S asignada en memoria* .

E/S asignada en memoria Método de direccionamiento de módulos de E/S y dispositivos externos. Se usa un único espacio de direcciones tanto para las direcciones de memoria principal como para E/S, y las mismas instrucciones máquina se utilizan tanto para lectura/escritura en memoria como para E/S.

E/S dirigida por interrupciones Tipo de E/S. La CPU da una orden de E/S, continua ejecutando las instrucciones siguientes, y es interrumpida por el módulo de E/S cuando ha terminado su trabajo.

E/S programada Una forma de E/S en la que la CPU envía una orden de E/S a un módulo de E/S y debe, por tanto, esperar a que la operación termine antes de proceder a concluir su ejecución.

Ejecución de predicados Mecanismo que permite la ejecución condicional de instrucciones individuales. Esto hace posible ejecutar especulativamente las distintas ramificaciones de una instrucción de ramificación y retener los resultados de la ramificación que definitivamente se seleccionó.

Ejecución especulativa Ejecución de instrucciones a lo largo de uno de los caminos de una bifurcación. Si al final resulta que no se ha tomado esta rama, entonces los resultados de la ejecución especulativa se descartan.

Emisión de instrucciones Proceso de iniciar la ejecución de una instrucción en las unidades funcionales del procesador. Esto sucede cuando una instrucción pasa de la fase de decodificación del cauce al estado de la primera ejecución del cauce.

Emulación Imitación de todo o parte de un sistema por otro, usualmente hardware, de forma que el sistema de imitación acepta los mismos datos, ejecuta los mismos programas, y produce los mismos resultados que el sistema imitado.

Entrada-salida (E/S) Se refiere a una entrada o una salida o a ambos. Trata de la transferencia de datos entre un computador y un periférico unido directamente a él.

Equipo periférico (IBM) En un computador, con respecto a una unidad de procesamiento particular, es cualquier equipo que proporcione a la unidad de procesamiento una comunicación externa. Sinónimo de *periférico.*

Escalar Cantidad caracterizada por un único valor.

Espacio de direcciones Rango de direcciones (memoria, E/S) que se pueden referenciar.

Fallo de página Se produce cuando se referencia una palabra que está en una página que no se encuentra en la memoria principal. Esto causa una interrupción y requiere que el sistema operativo proporcione la página que se necesita.

Firmware Microcódigo almacenado en memorias de solo lectura.

Formato de una instrucción Estructura de una instrucción como secuencia de bits. El formato divide la instrucción en campos, que corresponden a los elementos que constituyen la instrucción (por ejemplo, código de operación, operandos).

G Prefijo que indica *mil millones.*

Indexación Técnica de modificación de direcciones mediante registros índice.

Instrucción de un computador Instrucción que puede ser reconocida por la unidad de procesamiento del computador para la que fue diseñada. Sinónimo de *instrucción máquina.*

Interrupción deshabilitada Condición, normalmente creada por la CPU, durante la que la CPU ignora las señales de petición de interrupción de un tipo especificado.

Interrupción habilitada Condición, normalmente determinada por la CPU, mediante la cual la CPU responde a las señales de petición de interrupción de una clase predeterminada.

Interrupción Suspensión de un proceso (ejecución de un programa) causada por un evento externo, y realizada de tal forma que el proceso se puede reanudar.

K Prefijo que significa $2^{10} = 1024$. Entonces, 2 Kb $= 2 \times 2^{20}$ bits.

Lenguaje de microprogamación Conjunto de instrucciones usado para especificar microprogramas.

Lenguaje ensamblador Lenguaje orientado al computador cuyas instrucciones se suelen corresponder una a una con las instrucciones del computador y que posibilita poder definir, por ejemplo, macroinstrucciones. Es sinónimo de *lenguaje dependiente del computador.*

Línea de caché Bloque de datos asociado a una etiqueta de la caché y a la unidad de transferencia entre caché y memoria.

Localidad de referencia Tendencia de un procesador a acceder al mismo conjunto de lugares de memoria de forma repetida en periodo de tiempo corto.

M Prefijo que significa $2^{20} = 1.048.576$. Por tanto, 2 Mb $= 2 \times 2^{20}$ bits.

***Mainframe* o Computador grande** Término que se refería originalmente al chasis que contenía la unidad central de procesamiento o "*main frame*" de una máquina grande. Después del surgimiento de diseños de minicomputadores más pequeños a principios de los 70, las máquinas tradicionales grandes se describían como computadores *mainframe*. Las características típicas de un *mainframe* son: soportan grandes bases de datos, tienen un hardware de E/S sofisticado, y se usan en instalaciones centrales de procesamiento de datos.

Marco de página Zona de la memoria principal para guardar una página.

Maya en cadena Método de interconexión de dispositivos para determinar prioridades de interrupción conectando las fuentes de interrupción vía serie.

Memoria asociativa Memoria cuyas posiciones de almacenamiento se identifican por su contenido, o por parte de éste, en vez de por sus nombres o posiciones.

Memoria caché Buffer especial de almacenamiento, menor y más rápido que la memoria principal, que es usado para guardar una copia de las instrucciones y datos de la memoria principal que el procesador va a necesitar, y que obtiene automáticamente de la memoria principal.

Memoria de acceso aleatorio (RAM) Memoria en la que cada posición direccionable tiene un único mecanismo de direccionamiento. El tiempo de acceso a una posición dada es independiente de la secuencia de acceso previa.

Memoria de control Porción de la memoria que contiene microcódigo.

Memoria de sólo lectura (ROM) Memoria semiconductora cuyo contenido no se puede cambiar, salvo destruyéndola. Es una memoria no borrable.

Memoria no volátil Memoria cuyo contenido es estable y no requiere una fuente de alimentación permanente.

Memoria principal Memoria direccionable por programa en la que se pueden cargar, en posiciones, instrucciones y otros datos para su ejecución o procesamiento posterior.

Memoria programable de solo lectura (PROM) Memoria semiconductora cuyo contenido se puede establecer una sola vez. El proceso de escritura se lleva a cabo eléctricamente y lo puede hacer el usuario después de la fabricación del chip.

Memoria secundaria Memoria situada fuera del computador, como disco o cinta.

Memoria virtual Memoria que permite ser vista por el usuario como una memoria principal direccionable de un computador, en el que las direcciones virtuales se transforman en direcciones reales. El tamaño de la memoria virtual está limitado por el modo de direccionamiento del computador y por la cantidad de memoria exterior disponible y no por el número real de posiciones de la memoria principal.

Memoria volátil Memoria que requiere permanentemente alimentación eléctrica para mantener su contenido. Si la alimentación se interrumpe, la información se pierde.

Microcomputador Computador cuya unidad de procesamiento es un microprocesador. Un microcomputador básico incluye un microprocesador, una memoria, y entradas/salidas que pueden o no estar en un solo chip.

Microinstrucción Instrucción que controla el flujo de datos y el secuenciamiento en un procesador a un nivel más básico que las instrucciones máquina. Las instrucciones máquina individuales y quizás otras funciones se pueden implementar en microprogramas.

Microoperación Operación elemental de la CPU, ejecutada durante un pulso de reloj.

Microprocesador Un procesador cuyos elementos se han miniaturizado en uno o varios circuitos integrados.

Microprograma Secuencia de microinstrucciones que están en una memoria especial donde se puede acceder a ellas dinámicamente para realizar funciones diversas.

Módulo de E/S Uno de los principales componentes del computador. Es responsable del control de uno o más dispositivos externos (periféricos) y del intercambio de datos entre estos y la memoria principal y/o los registros de la CPU.

Monoprocesamiento Ejecución secuencial de instrucciones bajo la dirección de una unidad de procesamiento, o uso independiente de la unidad de procesamiento en un sistema multiprocesador.

Multiplexor Circuito combinacional que conecta una de las varias entradas a una sola salida. En un momento dado, sólo una de las entradas puede estar seleccionada para pasar a la salida.

Multiprocesador Computador que tiene uno o más procesadores con acceso común a la memoria principal.

Multiprocesador con acceso a memoria no uniforme (NUMA) Multiprocesador con memoria compartida en el que el tiempo de acceso a memoria desde un procesador dado a una palabra de memoria varía con la posición de la palabra en la memoria.

Multiprocesamiento simétrico (SMP) Forma de multiprocesamiento que permite a un sistema operativo ejecutar en cualquier procesador disponible o en un conjunto de procesadores disponibles simultáneamente.

Multiprogramación Forma de funcionamiento que posibilita la ejecución intercalada de dos o más programas, con un solo procesador.

Multitarea Forma de funcionamiento que posibilita la ejecución concurrente o intercalada de dos o más tareas. Lo mismo que multiprogramación, utilizando otra terminología.

Núcleo Parte del sistema operativo que contiene las funciones básicas y más usadas. A menudo, los núcleos permanecen residentes en la memoria principal.

Operador binario Operador que representa una operación con dos y solamente dos operandos.

Operador unitario Operador que representa una operación con un sólo operando.

Operando Entidad en la que se realiza una operación.

Ortogonalidad Principio por el que dos variables o dimensiones son independientes una de la otra. En el contexto de un conjunto de instrucciones, el término se usa generalmente para indicar que los elementos de una instrucción (modo de direccionamiento, número de operandos, longitud del operando) son independientes de (o no están determinados por) el código de operación.

Página En un sistema de memoria virtual, un bloque con una longitud fija que tiene una dirección virtual y que se transfiere como una unidad entre la memoria principal y la memoria auxiliar.

Palabra Conjunto ordenado de bytes o bits que determina la unidad normal en la que se almacena la información, transmitida, o con la que se opera en un computador dado. Normalmente, si el procesador tiene un conjunto de instrucciones de longitud fija, la longitud de una instrucción coincide con la de una palabra.

Palabra de estado de un programa (PSW) Un área en la memoria usada para indicar el orden en que se deben ejecutar las instrucciones, y mantiene e indica el estado del computador. Sinónimo de *palabra de estado del procesador.*

Palabra de instrucción muy larga (VLIW, "Very Long Instruction Word") Se refiere al uso de instrucciones que contienen operaciones múltiples. En efecto, las instrucciones múltiples están contenidas en una sola palabra. Usualmente, el compilador construye VLIW, situando las operaciones que se pueden ejecutar en paralelo en la misma palabra.

Paquete de discos Conjunto de discos magnéticos que se pueden sacar como un todo de la unidad de disco, junto con un contenedor desde el que el conjunto debe ser sacado cuando se utilice.

Pila Lista que se forma y mantiene de forma que el elemento próximo a extraer es el que se ha almacenado más recientemente, último en entrar, primero en salir (LIFO, "Last In First Out").

Predicción de saltos Mecanismo usado por el procesador para predecir el inicio de una rama de un programa previamente a su ejecución.

Procesador con supersegmentación de cauce Procesador en el que el cauce de segmentación de las instrucciones está formado por etapas muy pequeñas, de forma que se puede ejecutar más de una etapa en un ciclo de reloj, y por tanto, puede haber simultáneamente muchas instrucciones en el cauce.

Procesador de E/S Módulo de E/S con procesador propio, capaz de ejecutar instrucciones especializadas de E/S, o en algunos casos instrucciones máquina de uso general.

Procesador superescalar Procesador que incluye cauces segmentados para múltiples instrucciones, de forma que se pueden ejecutar simultáneamente más de una instrucción en la misma etapa.

Procesador Unidad funcional que interpreta y ejecuta instrucciones en un computador. Un procesador consta al menos de una unidad de control y una unidad aritmética.

Proceso Programa en ejecución. Un proceso se controla y temporiza con el sistema operativo.

Protocolo de coherencia de caché Mecanismo para mantener la validez de los datos entre varias cachés de forma que cada acceso a los datos adquirirá siempre la versión más reciente del contenido de una palabra de la memoria principal.

Puerta Circuito electrónico que produce una señal de salida que es una simple operación booleana de sus señales de entrada.

RAM dinámica RAM cuyas celdas se implementan usando condensadores. Una RAM dinámica pierde gradualmente sus datos salvo si se refresca periódicamente.

RAM estática RAM cuyas celdas se implementan con biestables. Una RAM estática mantendrá sus datos mientras se le suministre corriente; no se necesita un refresco periódico.

Registro búffer de memoria (MBR) Registro que contiene datos leídos de la memoria o datos a escribir en la memoria.

Registro de dirección de instrucción Registro de uso especial utilizado para guardar la dirección de la siguiente instrucción que se va a ejecutar.

Registro de dirección de memoria (MAR) Registro, de la unidad de procesamiento, que contiene la dirección de la posición de memoria a la que se va a acceder.

Registro de uso general Registro, normalmente direccionable explícitamente, dentro de un conjunto de ellos, que se puede usar para distintos objetivos, por ejemplo como acumulador, registro de indexación, o manipulador especial de datos.

Registro índice Registro cuyo contenido puede ser usado para modificar una dirección de un operando durante la ejecución de instrucciones; también puede ser usado como un contador. Un registro índice puede usarse para controlar la ejecución de un bucle, controlar el uso de un array, como conmutador, como índice de una tabla de consulta, o como un puntero.

Registro instrucción Registro que se usa para guardar e interpretar la instrucción a ejecutar.

Registros de control Registros de la CPU empleados para controlar operaciones de la CPU. La mayoría de estos registros no son visibles al usuario.

Registros Memoria muy rápida interna a la CPU. Algunos registros son visibles al usuario, es decir, se pueden programar a través del conjunto de instrucciones máquina. Otros registros sólo los puede usar la CPU, con fines de control.

Registros visibles al usuario Registros de la CPU que pueden ser referenciados por el programador. El formato del conjunto de instrucciones permite especificar uno o más registros como operandos o direcciones de operandos.

Repertorio de instrucciones de un computador Conjunto completo de instrucciones de un computador junto con una descripción de los significados que se pueden atribuir a las mismas. Sinónimo de *conjunto de instrucciones máquina.*

Representación en complemento a dos Se usa para representar enteros binarios. Un entero positivo se representa en signo y magnitud. Un número negativo se representa sumando uno a la representación en complemento a uno del mismo número.

Representación en complemento a uno Usada para representar enteros binarios. Un entero positivo se representa en signo y magnitud. Un entero negativo se representa invirtiendo cada bit en la representación de un entero positivo de la misma magnitud.

Representación en signo y magnitud Usada para representar enteros binarios. En una palabra de N bits, el bit que está más a la izquierda es el bit de signo ($0 =$ positivo, $1 =$ negativo) y el resto, los otros $N - 1$ bits, son la magnitud del número.

Salto condicional Salto que se produce sólo cuando la instrucción se ejecuta y además se satisfacen unas condiciones especiales. Contrastar con *salto incondicional.*

Salto incondicional Salto que se produce siempre que la instrucción que lo contiene se ejecute.

Segmentación de cauce Organización de un procesador en la que el procesador consta de una serie de etapas, permitiendo la ejecución concurrente de instrucciones múltiples.

Semiconductor Material semicristalino, como silicio o germanio, cuya conductividad eléctrica está entre aislante y buen conductor. Se usa para fabricar transistores y componentes de estado sólido.

Sistema de representación en coma fija Sistema de numeración en el que la posición de la coma que separa la parte entera de la decimal se ha fijado implícitamente en la serie de dígitos por alguna convención previamente establecida.

Sistema de representación en coma flotante Sistema de numeración en el que un número real es representado por un par de números, siendo el valor del número real el producto de uno de los números (que está representado en coma fija) por la base implícita elevada a la potencia indicada por el segundo número (que es el exponente).

Sistema operativo Software que controla la ejecución de programas y ofrece servicios como reserva de recursos, planificación, control de entradas/salidas, y gestión de datos.

Tabla de verdad Tabla que describe una función lógica mostrando todas la combinaciones posibles de las entradas e indicando, para cada combinación, la salida.

Temporización asíncrona Técnica en la que la aparición de un evento en el bus sigue y depende de la aparición de un evento previo.

Temporización síncrona Técnica en la que la aparición de un evento en el bus viene determinada por un reloj. El reloj define intervalos de tiempo de igual duración, y los eventos sólo pueden empezar al principio de un intervalo.

Tiempo de ciclo de memoria Inversa de la velocidad a la que se puede acceder a la memoria. Es el tiempo mínimo entre la respuesta a una petición de acceso (lectura o escritura) y la respuesta a la próxima petición de acceso.

Tiempo de ciclo del procesador Tiempo requerido para realizar la microoperación más corta de la CPU. Es la unidad básica de tiempo para medir todas las acciones de la CPU. Sinónimo de *tiempo de ciclo máquina.*

Unidad aritmético lógica (ALU) Parte del computador que realiza las operaciones aritméticas, lógicas y de relación.

Unidad central de procesamiento (CPU) Parte del computador que capta y ejecuta instrucciones. Está formada por la Unidad Aritmético Lógica (ALU), la unidad de control y los registros. A menudo se le suele llamar *procesador.*

Unidad de control Parte de la CPU que controla las operaciones de la CPU, incluyendo las operaciones de la ALU, las transferencias de datos en la CPU, y el intercambio de datos y señales de control a través de las interfaces externas (por ejemplo, a través del bus).

Variable global Variable definida en una parte de un programa que es usada al menos en alguna otra parte del programa.

Variable local Variable que se define y se usa sólo en una parte de un programa.

Vector Cantidad que se caracteriza por un conjunto ordenado de escalares.

REFERENCIAS

ABREVIATURAS

ACM Association for Computing Machinery
IBM International Business Machines Corporation
IEEE Institute of Electrical and Electronics Engineers

ABBO04 Abbot, D. *PCI Bus Demystified.* New York: Elsevier, 2004.

ACOS86 Acosta, R.; Kjelstrup, J.; and Torng, H. "An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors." *IEEE Transactions on Computers*, September 1986.

ADAM91 Adamek, J. *Foundations of Coding.* New York: Wiley, 1991.

AGAR89 Agarwal, A. *Analysis of Cache Performance for Operating Systems and Multiprogramming.* Boston: Kluwer Academic Publishers, 1989.

AGER87 Agerwala, T., and Cocke, J. *High Performance Reduced Instruction Set Processors.* Technical Report RC12434 (#55845). Yorktown, NY: IBM Thomas J. Watson Research Center, January 1987.

ANDE67a Anderson, D.; Sparacio, F.; and Tomasulo, F. "The IBM System/360 Model 91: Machine Philosophy and Instruction Handling." *IBM Journal of Research and Development*, January 1967.

ANDE67b Anderson, S., et al. "The IBM System/360 Model 91: Floating-Point Execution Unit." *IBM Journal of Research and Development*, January 1967. Reprinted in [SWAR90, Volume 1].

ANDE98 Anderson, D. *FireWire System Architecture.* Reading, MA: Addison-Wesley, 1998.

ANDE03 Anderson, D. "You Don't Know Jack About Disks." *ACM Queue*, June 2003.

ANTH04 Anthes, G. "CPUs Rev New Engines." *Computerworld*, October 25, 2004

ASH90 Ash, R. *Information Theory.* New York: Dover, 1990.

ATKI96 Atkins, M. "PC Software Performance Tuning." *IEEE Computer*, August 1996.

AZIM92 Azimi, M.; Prasad, B.; and Bhat, K. "Two Level Cache Architectures." *Proceedings COMPCON '92*, February 1992.

BAEN97 Baentsch, M., et al. "Enhancing the Web's Infrastructure: From Caching to Replication." *Internet Computing*, March/April 1997.

BAIL93 Bailey, D. "RISC Microprocessors and Scientific Computing." *Proceedings, Supercomputing '93*, 1993.

BASH81 Bashe, C.; Bucholtz, W.; Hawkins, G.; Ingram, J.; and Rochester, N. "The Architecture of IBM's Early Computers." *IBM Journal of Research and Development*, September 1981.

BASH91 Bashteen, A.; Lui, I.; and Mullan, J. "A Superpipeline Approach to the MIPS Architecture." *Proceedings, COMPCON Spring '91*, February 1991.

BELL70 Bell, C.; Cady, R.; McFarland, H.; Delagi, B.; O'Loughlin, J.; and Noonan, R. "A New Architecture for Minicomputers—The DEC PDP-11." *Proceedings, Spring Joint Computer Conference*, 1970.

BELL71a Bell, C., and Newell, A. *Computer Structures: Readings and Examples.* New York: McGraw-Hill, 1971.

BELL74 Bell, J.; Casasent, D.; and Bell, C. "An Investigation into Alternative Cache Organizations." *IEEE Transactions on Computers*, April 1974. http://research.microsoft.com/users/GBell/ gbvita.htm.

BELL78a Bell, C.; Mudge, J.; and McNamara, J. *Computer Engineering: A DEC View of Hardware Systems Design.* Bedford, MA: Digital Press, 1978.

BELL78b Bell, C.; Newell, A.; and Siewiorek, D. "Structural Levels of the PDP-8." in [BELL78a].

BELL78c Bell, C.; Kotok, A.; Hastings, T.; and Hill, R. "The Evolution of the DEC System-10." *Communications of the ACM*, January 1978.

BENH92 Benham, J. "A Geometric Approach to Presenting Computer Representations of Integers." *SIGCSE Bulletin*, December 1992.

BETK97 Betker, M.; Fernando, J.; and Whalen, S. "The History of the Microprocessor." *Bell Labs Technical Journal*, Autumn 1997.

BEZ03 Bez, R.; et al. Introduction to Flash Memory. *Proceedings of the IEEE*, April 2003.

BHAR00 Bharandwaj, J., et al. "The Intel IA-64 Compiler Code Generator." *IEEE Micro*, September/October 2000.

BLAA97 Blaauw, G., and Brooks, F. *Computer Architecture: Concepts and Evolution*. Reading, MA: Addison-Wesley, 1997.

BLAH83 Blahut, R. *Theory and Practice of Error Control Codes.* Reading, MA: Addison-Wesley, 1983.

BOHR98 Bohr, M. "Silicon Trends and Limits for Advanced Microprocessors." *Communications of the ACM*, March 1998.

BOHR03 Bohr, M. "High Performance Logic Technology and Reliability Challenges." *International Reliability Physics Symposium*, March 2003. http://www.irps.org/03-41st

BORK03 Borkar, S. "Getting Gigascale Chips: Challenges and Opportunities in Continuing Moore's Law." *ACM Queue*, October 2003.

BRAD91a Bradlee, D.; Eggers, S.; and Henry, R. "The Effect on RISC Performance of Register Set Size and Structure Versus Code Generation Strategy." *Proceedings, 18th Annual International Symposium on Computer Architecture*, May 1991.

BRAD91b Bradlee, D.; Eggers, S.; and Henry, R. "Integrating Register Allocation and Instruction Scheduling for RISCs." *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.

BREW97 Brewer, E. "Clustering: Multiply and Conquer." *Data Communications*, July 1997.

BREY03 Brey, B. *The Intel Microprocessors: 8086/8066, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4.* Upper Saddle River, NJ: Prentice Hall, 2000.

BURG97 Burger, D., and Austin, T. "The SimpleScalar Tool Set, Version 2.0." *Computer Architecture News*, June 1997.

BURK46 Burks, A.; Goldstine, H.; and von Neumann, J. *Preliminary Discussion of the Logical Design of an Electronic Computer Instrument*. Report prepared for U.S. Army Ordnance Dept., 1946, reprinted in [BELL71a].

BUYY99a Buyya, R. *High Performance Cluster Computing: Architectures and Systems.* Upper Saddle River, NJ: Prentice Hall, 1999.

BUYY99b Buyya, R. *High Performance Cluster Computing: Programming and Applications.* Upper Saddle River, NJ: Prentice Hall, 1999.

CART96 Carter, J. *Microprocesser Architecture and Microprogramming.* Upper Saddle River, NJ: Prentice Hall, 1996.

CATA94 Catanzaro, B. *Multiprocessor System Architectures.* Mountain View, CA: Sunsoft Press, 1994.

CHAI82 Chaitin, G. "Register Allocation and Spilling via Graph Coloring." *Proceedings, SIGPLAN Symposium on Compiler Construction*, June 1982.

CHAS00 Chasin, A. "Predication, Speculation, and Modern CPUs." *Dr. Dobb's Journal*, May 2000.

CHEN94 Chen, P.; Lee, E.; Gibson, G.; Katz, R.; and Patterson, D. "RAID: High-Performance, Reliable Secondary Storage." *ACM Computing Surveys*, June 1994.

CHEN96 Chen, S., and Towsley, D. "A Performance Evaluation of RAID Architectures." *IEEE Transactions on Computers*, October 1996.

CHOW86 Chow, F.; Himmelstein, M.; Killian, E.; and Weber, L. "Engineering a RISC Compiler System." *Proceedings, COMPCON Spring '86*, March 1986.

CHOW87 Chow, F.; Correll, S.; Himmelstein, M.; Killian, E.; and Weber, L. "How Many Addressing Modes Are Enough?" *Proceedings, Second International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1987.

CHOW90 Chow, F., and Hennessy, J. "The Priority-Based Coloring Approach to Register Allocation." *ACM Transactions on Programming Languages*, October 1990.

CLAR85 Clark, D., and Emer, J. "Performance of the VAX-11/780 Translation Buffer: Simulation and Measurement." *ACM Transactions on Computer Systems*, February 1985.

CLEM00 Clements, A. "The Undergraduate Curriculum in Computer Architecture." *IEEE Micro*, May/June 2000.

COHE81 Cohen, D. "On Holy Wars and a Plea for Peace." *Computer*, October 1981.

COLW85a Colwell, R.; Hitchcock, C.; Jensen, E.; Brinkley-Sprunt, H.; and Kollar, C. "Computers, Complexity, and Controversy." *Computer*, September 1985.

COLW85b Colwell, R.; Hitchcock, C.; Jensen, E.; and Sprunt, H. "More Controversy About 'Computers, Complexity, and Controversy."' *Computer,* December 1985.

COME95 Comerford, R. "An Overview of High Performance." *IEEE Spectrum*, April 1995.

COME00 Comerford, R. "Magnetic Storage: The Medium that Wouldn't Die." *IEEE Spectrum*, December 2000.

COOK82 Cook, R., and Dande, N. "An Experiment to Improve Operand Addressing." *Proceedings, Symposium on Architecture Support for Programming Languages and Operating Systems*, March 1982.

COON81 Coonen J. "Underflow and Denormalized Numbers." *IEEE Computer,* March 1981.

COUT86 Coutant, D.; Hammond, C.; and Kelley, J. "Compilers for the New Generation of Hewlett-Packard Computers." *Proceedings, COMPCON Spring '86,* March 1986.

CRAG79 Cragon, H. "An Evaluation of Code Space Requirements and Performance of Various Architectures." *Computer Architecture News*, February 1979.

CRAG92 Cragon, H. *Branch Strategy Taxonomy and Performance Models*. Los Alamitos, CA: IEEE Computer Society Press, 1992.

CRAW90 Crawford, J. "The i486 CPU: Executing Instructions in One Clock Cycle." *IEEE Micro,* February 1990.

CRIS97 Crisp, R. "Direct RAMBUS Technology: The New Main Memory Standard." *IEEE Micro*, November/December 1997.

CUPP01 Cuppu, V., et al. "High Performance DRAMS in Workstation Environments." *IEEE Transactions on Computers,* November 2001.

DATT93 Dattatreya, G. "A Systematic Approach to Teaching Binary Arithmetic in a First Course." *IEEE Transactions on Education*, February 1993.

DAVI87 Davidson, J., and Vaughan, R. "The Effect of Instruction Set Complexity on Program Size and Memory Performance." *Proceedings, Second International Conference on Architectural Support for Programming Languages and Operating Systems,* October 1987.

DENN68 Denning, P. "The Working Set Model for Program Behavior." *Communications of the ACM,* May 1968.

DERO87 DeRosa, J., and Levy, H. "An Evaluation of Branch Architectures." *Proceedings, Fourteenth Annual International Symposium on Computer Architecture*, 1987.

DEWA90 Dewar, R., and Smosna, M. *Microprocessors: A Programmer's View.* New York: McGraw-Hill, 1990.

DIEF94 Diefendorff, K. "History of the PowerPC Architecture." *Communications of the ACM,* June 1994.

DIJK63 Dijkstra, E. "Making an ALGOL Translator for the X1." In *Annual Review of Automatic Programming, Volume 4.* Pergamon, 1963.

DOWD98 Dowd, K., and Severance, C. *High Performance Computing.* Sebastopol, CA: O'Reilly, 1998.

DUBE91 Dubey, P., and Flynn, M. "Branch Strategies: Modeling and Optimization." *IEEE Transactions on Computers,* October 1991.

DULO98 Dulong, C. "The IA-64 Architecture at Work." *Computer,* July 1998.

ECKE90 Eckert, R. "Communication Between Computers and Peripheral Devices—An Analogy." *ACM SIGCSE Bulletin*, September 1990.

ERCE04 Ercegovac, M., and Lang, T. *Digital Arithmetic.* San Francisco: Morgan Kaufmann, 2004.

ELAY85 El-Ayat, K., and Agarwal, R. "The Intel 80386—Architecture and Implementation." *IEEE Micro,* December 1985.

EVAN03 Evans, J., and Trimper, G. *Itanium Architecture for Programmers.* Upper Saddle River, NJ: Prentice Hall, 2003.

EVEN00a Even, G., and Paul, W. "On the Design of IEEE Compliant Floating-Point Units." *IEEE Transactions on Computers,* May 2000.

EVEN00b Even, G., and Seidel, P. "A Comparison of Three Rounding Algorithms for IEEE Floating-Point Multiplication." *IEEE Transactions on Computers,* July 2000.

EVER98 Evers, M., et al. "An Analysis of Correlation and Predictability: What Makes Two-Level Branch Predictors Work." *Proceedings, 25th Annual International Symposium on Microarchitecture,* July 1998.

EVER01 Evers, M., and Yeh, T. "Understanding Branches and Designing Branch Predictors for High-Performance Microprocessors." *Proceedings of the IEEE,* November 2001.

FARH04 Farhat, H. *Digital Design and Computer Organization.* Boca Raton, FL: CRC Press, 2004.

FARM92 Farmwald, M., and Mooring, D. "A Fast Path to One Memory." *IEEE Spectrum,* October 1992.

FLYN72 Flynn, M. "Some Computer Organizations and Their Effectiveness." *IEEE Transactions on Computers,* September 1972.

FLYN85 Flynn, M.; Johnson, J.; and Wakefield, S. "On Instruction Sets and Their Formats." *IEEE Transactions on Computers,* March 1985.

FLYN87 Flynn, M.; Mitchell, C.; and Mulder, J. "And Now a Case for More Complex Instruction Sets." *Computer,* September 1987.

FLYN01 Flynn, M., and Oberman, S. *Advanced Computer Arithmetic Design.* New York: Wiley, 2001.

FRAI83 Frailey, D. "Word Length of a Computer Architecture: Definitions and Applications." *Computer Architecture News,* June 1983.

FRIE96 Friedman, M. "RAID Keeps Going and Going and . . ." *IEEE Spectrum,* April 1996.

FURH87 Furht, B., and Milutinovic, V. "A Survey of Microprocessor Architectures for Memory Management." *Computer,* March 1987.

FUTR01 Futral, W. *InfiniBand Architecture: Development and Deployment.* Hillsboro, OR: Intel Press, 2001.

GHAI98 Ghai, S.; Joyner, J.; and John, L. *Investigating the Effectiveness of a Third Level Cache.* Technical Report TR-980501-01, Laboratory for Computer Architecture, University of Texas at Austin. http://www.ece.utexas.edu/projects/ece/lca/publications.html

GIBB04 Gibbs, W. "A Split at the Core." *Scientific American,* November 2004.

GIFF87 Gifford, D., and Spector, A. "Case Study: IBM's System/360-370 Architecture." *Communications of the ACM,* April 1987.

GOLD91 Goldberg, D. "What Every Computer Scientist Should Know About Floating-Point Arithmetic." *ACM Computing Surveys,* March 1991.

GOOD83 Goodman, J. "Using Cache Memory to Reduce Processor-Memory Bandwidth." *Proceedings, 10th Annual International Symposium on Computer Architecture,* 1983. Reprinted in [HILL00].

GREG98 Gregg, J. *Ones and Zeros: Understanding Boolean Algebra, Digital Circuits, and the Logic of Sets.* New York: Wiley, 1998.

HAMM97 Hammond, L.; Nayfay, B.; and Olukotun, K. "A Single-Chip Multiprocessor." *Computer,* September 1997.

HAND98 Handy, J. *The Cache Memory Book.* San Diego: Academic Press, 1993.

HALF97 Halfhill, T. "Beyond Pentium II." *Byte,* December 1997.

HAYE98 Hayes, J. *Computer Architecture and Organization.* New York: McGraw-Hill, 1998.

HEAT84 Heath, J. "Re-evaluation of RISC I." *Computer Architecture News,* March 1984.

HENN82 Hennessy, J., et al. "Hardware/Software Tradeoffs for Increased Performance." *Proceedings, Symposium on Architectural Support for Programming Languages and Operating Systems,* March 1982.

HENN84 Hennessy, J. "VLSI Processor Architecture." *IEEE Transactions on Computers,* December 1984.

HENN91 Hennessy, J., and Jouppi, N. "Computer Technology and Architecture: An Evolving Interaction." *Computer,* September 1991.

HIDA90 Hidaka, H.; Matsuda, Y.; Asakura, M.; and Kazuyasu, F. "The Cache DRAM Architecture: A DRAM with an On-Chip Cache Memory." *IEEE Micro,* April 1990.

HIGB90 Higbie, L. "Quick and Easy Cache Performance Analysis." *Computer Architecture News,* June 1990.

HILL64 Hill, R. "Stored Logic Programming and Applications." *Datamation,* February 1964.

HILL89 Hill, M. "Evaluating Associativity in CPU Caches." *IEEE Transactions on Computers,* December 1989.

HILL00 Hill, M.; Jouppi, N.; and Sohi, G. *Readings in Computer Architecture.* San Francisco: Morgan Kaufmann, 2000.

HINT01 Hinton, G., et al. "The Microarchitecture of the Pentium 4 Processor." *Intel Technology Journal,* Q1 2001. http://developer.intel.com/technology/itj/

HUCK83 Huck, T. *Comparative Analysis of Computer Architectures.* Stanford University Technical Report No. 83-243, May 1983.

HUCK00 Huck, J., et al. "Introducing the IA-64 Architecture." *IEEE Micro,* September/October 2000.

HUGU91 Huguet, M., and Lang, T. "Architectural Support for Reduced Register Saving/Restoring in Single-Window Register Files." *ACM Transactions on Computer Systems,* February 1991.

HUTC96 Hutcheson, G., and Hutcheson, J. "Technology and Economics in the Semiconductor Industry." *Scientific American,* January 1996.

HWAN93 Hwang, K. *Advanced Computer Architecture.* New York: McGraw-Hill, 1993.

HWAN99 Hwang, K, et al. "Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space." *IEEE Concurrency,* January-March 1999.

HWU98 Hwu, W. "Introduction to Predicated Execution." *Computer,* January 1998.

HWU01 Hwu, W.; August, D.; and Sias, J. "Program Decision Logic Optimization Using Predication and Control Speculation." *Proceedings of the IEEE*, November 2001.

IBM94 International Business Machines, Inc. *The PowerPC Architecture: A Specification for a New Family of RISC Processors.* San Francisco, CA: Morgan Kaufmann, 1994.

IBM01 International Business Machines, Inc. *64 Mb Synchronous DRAM.* IBM Data Sheet 364164, January 2001.

INTE98 Intel Corp. *Pentium Pro and Pentium II Processors and Related Products.* Aurora, CO, 1998.

INTE00a Intel Corp. *Intel IA-64 Architecture Software Developer's Manual (4 volumes).* Document 245317 through 245320. Aurora, CO, 2000.

INTE00b Intel Corp. *Itanium Processor Microarchitecture Reference for Software Optimization*. Aurora, CO, Document 245473. August 2000.

INTE01a Intel Corp. *Intel Pentium 4 Processor Optimization Reference Manual*. Document 248966-04 2001. http://developer.intel.com/design/Pentium4/documentation.htm.

INTE01b Intel Corp. *Desktop Performance and Optimization for Intel Pentium 4 Processor.* Document 248966-04 2001 http://developer.intel.com/design/Pentium4/documentation.htm.

INTE02 Intel Corp. *Intel Itanium Architecture Software Developer's Manual (3 volumes)*. Aurora, CO, 2002.

INTE04a Intel Corp. *IA-32 Intel Architecture Software Developer's Manual (3 volumes).* Aurora, CO, 2004.

INTE04b Intel Research and Development. *Architecting the Era of Tera*. Intel White Paper, February 2004. http://www.intel.com/labs/teraera/index.htm.

JAME90 James, D. "Multiplexed Buses: The Endian Wars Continue." *IEEE Micro,* September 1983.

JARP01 Jarp, S. "Optimizing IA-64 Performance." *Dr. Dobb's Journal*, July 2001.

JOHN91 Johnson, M. *Superscalar Microprocessor Design.* Englewood Cliffs, NJ: Prentice Hall, 1991.

JOUP88 Jouppi, N. "Superscalar versus Superpipelined Machines." *Computer Architecture News*, June 1988.

JOUP89a Jouppi, N., and Wall, D. "Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines." *Proceedings, Third International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1989.

JOUP89b Jouppi, N. "The Nonuniform Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance." *IEEE Transactions on Computers,* December 1989.

JTF01 Joint Task Force on Computing Curricula. *Computing Curricula 2001 Computer Science.* IEEE Computer Society and ACM, December 2001.

KAEL91 Kaeli, D., and Emma, P. "Branch History Table Prediction of Moving Target Branches Due to Subroutine Returns." *Proceedings, 18th Annual International Symposium on Computer Architecture*, May 1991.

KAGA01 Kagan, M. "InfiniBand: Thinking Outside the Box Design." *Communications System Design,* September 2001. (www.csdmag.com)

KALL04 Kalla, R.; Sinharoy, B.; and Tendler, J. "IBM Power5 Chip: A Dual-Core Multithreaded Processor." *IEEE Micro,* March-April 2004.

KANE92 Kane, G., and Heinrich, J. *MIPS RISC Architecture.* Englewood Cliffs, NJ: Prentice Hall, 1992.

KAPP00 Kapp, C. "Managing Cluster Computers." *Dr. Dobb's Journal,* July 2000.

KATE83 Katevenis, M. *Reduced Instruction Set Computer Architectures for VLSI*. Ph.D. dissertation, Computer Science Department, University of California at Berkeley, October 1983. Reprinted by MIT Press, Cambridge, MA, 1985.

KATH01 Kathail. B.; Schlansker, M.; and Rau, B. "Compiling for EPIC Architectures." *Proceedings of the IEEE*, November 2001.

KATZ89 Katz, R.; Gibson, G.; and Patterson, D. "Disk System Architecture for High Performance Computing." *Proceedings of the IEEE,* December 1989.

KEET01 Keeth, B., and Baker, R. *DRAM Circuit Design: A Tutorial.* Piscataway, NJ: IEEE Press, 2001.

KHUR01 Khurshudov, A. *The Essential Guide to Computer Data Storage.* Upper Saddle River, NJ: Prentice Hall, 2001.

KNUT71 Knuth, D. "An Empirical Study of FORTRAN Programs." *Software Practice and Experience,* vol. 1, 1971.

KNUT98 Knuth, D. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Reading, MA: Addison-Wesley, 1998.

KUCK77 Kuck, D.; Parker, D.; and Sameh, A. "An Analysis of Rounding Methods in Floating-Point Arithmetic." *IEEE Transactions on Computers.* July 1977.

KUGA91 Kuga, M.; Murakami, K.; and Tomita, S. "DSNS (Dynamically-hazard resolved, Statically-code-scheduled, Nonuniform Superscalar): Yet Another Superscalar Processor Architecture." *Computer Architecture News,* June 1991.

LEE91 Lee, R.; Kwok, A.; and Briggs, F. "The Floating Point Performance of a Superscalar SPARC Processor." *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.

LILJ88 Lilja, D. "Reducing the Branch Penalty in Pipelined Processors." *Computer,* July 1988.

LILJ93 Lilja, D. "Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons." *ACM Computing Surveys*, September 1993.

LOVE96 Lovett, T., and Clapp, R. "Implementation and Performance of a CC-NUMA System." *Proceedings, 23rd Annual International Symposium on Computer Architecture,* May 1996.

LUND77 Lunde, A. "Empirical Evaluation of Some Features of Instruction Set Processor Architectures." *Communications of the ACM,* March 1977.

LYNC93 Lynch, M. *Microprogrammed State Machine Design.* Boca Raton, FL: CRC Press, 1993.

MACD84 MacDougall, M. "Instruction-level Program and Process Modeling." *IEEE Computer*, July 1984.

MAHL94 Mahlke, S., et al. "Characterizing the Impact of Predicated Execution on Branch Prediction." *Proceedings, 27th International Symposium on Microarchitecture*, December 1994.

MAHL95 Mahlke, S., et al. "A Comparison of Full and Partial Predicated Execution Support for ILP Processors." *Proceedings, 22nd International Symposium on Computer Architecture,* June 1995.

MAK97 Mak, P., et al. "Shared-Cache Clusters in a System with a Fully Shared Memory." *IBM Journal of Research and Development,* July/September 1997.

MAK04 Mak, P., et al. "Processor Subsystem Interconnect for a Large Symmetric Multiprocessing System." *IBM Journal of Research and Development*, May/July 2004.

MANJ01a Manjikian, N. "More Enhancements of the SimpleScalar Tool Set." *Computer Architecture News*, September 2001.

MANJ01b Manjikian, N. "Multiprocessor Enhancements of the SimpleScalar Tool Set." *Computer Architecture News,* March 2001.

MANO04 Mano, M. *Logic and Computer Design Fundamentals.* Upper Saddle River, NJ: Prentice Hall, 2004.

MANS97 Mansuripur, M., and Sincerbox, G. "Principles and Techniques of Optical Data Storage." *Proceedings of the IEEE*, November 1997.

MARC90 Marchant, A. *Optical Recording.* Reading, MA: Addison-Wesley, 1990.

MARK00 Markstein, P. *IA-64 and Elementary Functions.* Upper Saddle River, NJ: Prentice Hall PTR, 2000.

MARR02 Marr, D.; et al. "Hyper-Threading Technology Architecture and Microarchitecture." *Intel Technology Journal*, First Quarter, 2002.

MASH95 Mashey, J. "CISC vs. RISC (or what is RISC really)." *USENET comp.arch newsgroup, article 46782,* February 1995.

MASS97 Massiglia, P. *The RAID Book: A Storage System Technology Handbook.* St. Peter, MN: The Raid Advisory Board, 1997.

MAYB84 Mayberry, W., and Efland, G. "Cache Boosts Multiprocessor Performance." *Computer Design,* November 1984.

MAZI03 Mazidi, M., and Mazidi, J. *The 80x86 IBM PC and Compatible Computers: Assembly Language, Design and Interfacing.* Upper Saddle River, NJ: Prentice Hall, 2003.

MCEL85 McEliece, R. "The Reliability of Computer Memories." *Scientific American,* January 1985.

MCNA03 McNairy, C., and Soltis, D. "Itanium 2 Processor Microarchitecture." *IEEE Micro,* March-April 2003.

MEE96a Mee, C., and Daniel, E. eds. *Magnetic Recording Technology.* New York: McGraw-Hill, 1996.

MEE96b Mee, C., and Daniel, E. eds. *Magnetic Storage Handbook.* New York: McGraw-Hill, 1996.

MILE00 Milenkovic, A. "Achieving High Performance in Bus-Based Shared-Memory Multiprocessors." *IEEE Concurrency,* July-September 2000.

MIRA92 Mirapuri, S.; Woodacre, M.; and Vasseghi, N. "The MIPS R4000 Processor." *IEEE Micro,* April 1992.

MOOR65 Moore, G. "Cramming More Components Onto Integrated Circuits." *Electronics Magazine*, April 19, 1965.

MORS78 Morse, S.; Pohlman, W.; and Ravenel, B. "The Intel 8086 Microprocessor: A 16-bit Evolution of the 8080." *Computer,* June 1978.

MOSH01 Moshovos, A., and Sohi, G. "Microarchitectural Innovations: Boosting Microprocessor Performance Beyond Semiconductor Technology Scaling." *Proceedings of the IEEE,* November 2001.

MYER78 Myers, G. "The Evaluation of Expressions in a Storage-to-Storage Architecture." *Computer Architecture News,* June 1978.

NAFF02 Naffziger, S., et al. "The Implementation of the Itanium 2 Microprocessor." *IEEE Journal of Solid-State Circuits,* November 2002.

NOVI93 Novitsky, J.; Azimi, M.; and Ghaznavi, R. "Optimizing Systems Performance Based on Pentium Processors." *Proceedings COMPCON '92,* February 1993.

OBER97a Oberman, S., and Flynn, M. "Design Issues in Division and Other Floating-Point Operations." *IEEE Transactions on Computers,* February 1997.

OBER97b Oberman, S., and Flynn, M. "Division Algorithms and Implementations." *IEEE Transactions on Computers,* August 1997.

OLUK96 Olukotun, K., et al. "The Case for a Single-Chip Multiprocessor." *Proceedings, Seventh International Conference on Architectural Support for Programming Languages and Operating Systems,* 1996.

OMON99 Omondi, A. *The Microarchitecture of Pipelined and Superscalar Computers.* Boston: Kluwer, 1999.

OVER01 Overton, M. *Numerical Computing with IEEE Floating Point Arithmetic.* Philadelphia, PA: Society for Industrial and Applied Mathematics, 2001.

PADE81 Padegs, A. "System/360 and Beyond." *IBM Journal of Research and Development*, September 1981.

PADE88 Padegs, A.; Moore, B.; Smith, R.; and Buchholz, W. "The IBM System/370 Vector Architecture: Design Considerations." *IEEE Transactions on Communications,* May 1988.

PARH00 Parhami, B. *Computer Arithmetic: Algorithms and Hardware Design.* Oxford: Oxford University Press, 2000.

PARK89 Parker, A., and Hamblen, J. *An Introduction to Microprogramming with Exercises Designed for the Texas Instruments SN74ACT8800 Software Development Board.* Dallas, TX: Texas Instruments, 1989.

PATT82a Patterson, D., and Sequin, C. "A VLSI RISC." *Computer,* September 1982.

PATT82b Patterson, D., and Piepho, R. "Assessing RISCs in High-Level Language Support." *IEEE Micro,* November 1982.

PATT84 Patterson, D. "RISC Watch." *Computer Architecture News,* March 1984.

PATT85a Patterson, D. "Reduced Instruction Set Computers." *Communications of the ACM.* January 1985.

PATT85b Patterson, D., and Hennessy, J. "Response to 'Computers, Complexity, and Controversy."' *Computer,* November 1985.

PATT88 Patterson, D.; Gibson, G.; and Katz, R. "A Case for Redundant Arrays of Inexpensive Disks (RAID)." *Proceedings, ACM SIGMOD Conference of Management of Data,* June 1988.

PATT01 Patt, Y. "Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution." *Proceedings of the IEEE,* November 2001.

PEIR99 Peir, J.; Hsu, W.; and Smith, A. "Functional Implementation Techniques for CPU Cache Memories." *IEEE Transactions on Computers,* February 1999.

PELE97 Peleg, A.; Wilkie, S.; and Weiser, U. "Intel MMX for Multimedia PCs." *Communications of the ACM,* January 1997.

PFIS98 Pfister, G. *In Search of Clusters.* Upper Saddle River, NJ: Prentice Hall, 1998.

POPE91 Popescu, V., et al. "The Metaflow Architecture." *IEEE Micro,* June 1991.

POTT94 Potter, T., et al. "Resolution of Data and Control-Flow Dependencies in the PowerPC 601." *IEEE Micro,* October 1994.

PRES01 Pressel, D. "Fundamental Limitations on the Use of Prefetching and Stream Buffers for Scientific Applications." *Proceedings, ACM Symposium on Applied Computing,* March 2001.

PRIN97 Prince, B. *Semiconductor Memories.* New York: Wiley, 1997.

PRIN02 Prince, B. *Emerging Memories: Technologies and Trends.* Norwell, MA: Kluwer, 2002.

PROT88 Protopapas, D. *Microcomputer Hardware Design.* Englewood Cliffs, NJ: Prentice Hall, 1988.

PRZY88 Przybylski, S.; Horowitz, M.; and Hennessy, J. "Performance Trade-Offs in Cache Design." *Proceedings, Fifteenth Annual International Symposium on Computer Architecture,* June 1988.

PRZY90 Przybylski, S. "The Performance Impact of Block Size and Fetch Strategies." *Proceedings, 17th Annual International Symposium on Computer Architecture,* May 1990.

RADI83 Radin, G. "The 801 Minicomputer." *IBM Journal of Research and Development,* May 1983.

RAGA83 Ragan-Kelley, R., and Clark, R. "Applying RISC Theory to a Large Computer." *Computer Design,* November 1983.

RECH98 Reches, S., and Weiss, S. "Implementation and Analysis of Path History in Dynamic Branch Prediction Schemes." *IEEE Transactions on Computers,* August 1998.

RODR01 Rodriguez, M.; Perez, J.; and Pulido, J. "An Educational Tool for Testing Caches on Symmetric Multiprocessors." *Microprocessors and Microsystems,* June 2001.

ROSC03 Rosch, W. *Winn L. Rosch Hardware Bible.* Indianapolis, IN: Que Publishing, 2003.

SAKA02 Sakai, S. "CMP on SoC: architect's view." Proceedings. 15th International Symposium on System Synthesis, 2002.

SATY81 Satyanarayanan, M., and Bhandarkar, D. "Design Trade-Offs in VAX-11 Translation Buffer Organization." *Computer,* December 1981.

SCHA97 Schaller, R. "Moore's Law: Past, Present, and Future." *IEEE Spectrum,* June 1997.

SCHL00a Schlansker, M.; and Rau, B. "EPIC: Explicitly Parallel Instruction Computing." *Computer,* February 2000.

SCHL00b Schlansker, M.; and Rau, B. *EPIC: An Architecture for Instruction-Level Parallel Processors.* HPL Technical Report HPL-1999-111, Hewlett-Packard Laboratories (www.hpl. hp.com), February 2000.

SCHW99 Schwarz, E., and Krygowski, C. "The S/390 G5 Floating-Point Unit." *IBM Journal of Research and Development,* September/November 1999.

SEBE76 Sebern, M. "A Minicomputer-compatible Microcomputer System: The DEC LSI-11." *Proceedings of the IEEE,* June 1976.

SEGE91 Segee, B., and Field, J. *Microprogramming and Computer Architecture.* New York: Wiley, 1991.

SERL86 Serlin, O. "MIPS, Dhrystones, and Other Tales." *Datamation,* June 1, 1986.

SHAN95 Shanley, T. *PowerPC System Architecture.* Reading, MA: Addison-Wesley, 1995.

SHAN98 Shanley, T. *Pentium Pro and Pentium II System Architecture.* Reading, MA: Addison-Wesley, 1998.

SHAN99 Shanley, T., and Anderson, D. *PCI Systems Architecture.* Richardson, TX: Mindshare Press, 1999.

SHAN03 Shanley, T. *InfinBand Network Architecture.* Reading, MA: Addison-Wesley, 2003.

SHAN05 Shanley, T. *Unabridged Pentium 4, The: IA32 Processor Genealogy.* Reading, MA: Addison-Wesley, 2005.

SHAR97 Sharma, A. *Semiconductor Memories: Technology, Testing, and Reliability.* New York: IEEE Press, 1997.

SHAR00 Sharangpani, H., and Arona, K. "Itanium Processor Microarchitecture." *IEEE Micro,* September/October 2000.

SHAR03 Sharma, A. *Advanced Semiconductor Memories: Architectures, Designs, and Applications.* New York: IEEE Press, 2003.

SHEN05 Shen, J., and Lipasti, M. *Modern Processor Design: Fundamentals of Superscalar Processors.* New York: McGraw-Hill, 2005.

SIEG04 Siegel, T.; Pfeffer, E.; and Magee, A. "The IBM z990 Microprocessor." *IBM Journal of Research and Development,* May/July 2004.

SIEW82 Siewiorek, D.; Bell, C.; and Newell, A. *Computer Structures: Principles and Examples.* New York: McGraw-Hill, 1982.

SIMA97 Sima, D. "Superscalar Instruction Issue." *IEEE Micro,* September/October 1997.

SIMA04 Sima, D. "Decisive Aspects in the Evolution of Microprocessors." *Proceedings of the IEEE,* December 2004.

SIMO96 Simon, H. *The Sciences of the Artificial.* Cambridge, MA: MIT Press, 1996.

SMIT82 Smith, A. "Cache Memories." *ACM Computing Surveys,* September 1992.

SMIT87 Smith, A. "Line (Block) Size Choice for CPU Cache Memories." *IEEE Transactions on Communications*, September 1987.

SMIT89 Smith, M.; Johnson, M.; and Horowitz, M. "Limits on Multiple Instruction Issue." *Proceedings, Third International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1989.

SMIT95 Smith, J., and Sohi, G. "The Microarchitecture of Superscalar Processors." *Proceedings of the IEEE*, December 1995.

SODE96 Soderquist, P., and Leeser, M. "Area and Performance Tradeoffs in Floating-Point Divide and Square-Root Implementations." *ACM Computing Surveys*, September 1996.

SOHI90 Sohi, G. "Instruction Issue Logic for High-Performance Interruptable, Multiple Functional Unit, Pipelined Computers." *IEEE Transactions on Computers*, March 1990.

STAL88 Stallings, W. "Reduced Instruction Set Computer Architecture." *Proceedings of the IEEE*, January 1988.

STAL04 Stallings, W. *Data and Computer Communications*, *Seventh Edition.* Upper Saddle River, NJ: Prentice Hall, 2004.

STAL05 Stallings, W. *Operating Systems, Internals and Design Principles, Fifth Edition.* Upper Saddle River, NJ: Prentice Hall, 2005.

STEN90 Stenstrom, P. "A Survey of Cache Coherence Schemes of Multiprocessors." *Computer,* June 1990.

STEV64 Stevens, W. "The Structure of System/360, Part II: System Implementation." *IBM Systems Journal*, Vol. 3, No. 2, 1964. Reprinted in [SIEW82].

STON93 Stone, H. *High-Performance Computer Architecture.* Reading, MA: Addison-Wesley, 1993.

STON96 Stonham, T. *Digital Logic Techniques.* London: Chapman & Hall, 1996.

STRE78 Strecker, W. "VAX-11/780: A Virtual Address Extension to the DEC PDP-11 Family." *Proceedings, National Computer Conference*, 1978.

STRE83 Strecker, W. "Transient Behavior of Cache Memories." *ACM Transactions on Computer Systems*, November 1983.

STRI79 Stritter, E., and Gunter, T. "A Microprocessor Architecture for a Changing World: The Motorola 68000." *Computer,* February 1979.

SWAR90 Swartzlander, E., editor. *Computer Arithmetic, Volumes I and II*. Los Alamitos, CA: IEEE Computer Society Press, 1990.

TAMI83 Tamir, Y., and Sequin, C. "Strategies for Managing the Register File in RISC." *IEEE Transactions on Computers,* November 1983.

TANE78 Tanenbaum, A. "Implications of Structured Programming for Machine Architecture." *Communications of the ACM*, March 1978.

TANE97 Tanenbaum, A., and Woodhull, A. *Operating Systems: Design and Implementation.* Upper Saddle River, NJ: Prentice Hall, 1997.

TEND02 Tendler, J., et al. "POWER4 System Microarchitecture." *IBM Journal of Research and Development*, January 2002.

THOM94 Thompson, T., and Ryan, B. "PowerPC 620 Soars." *Byte*, November 1994.

THOM00 Thompson, D. "IEEE 1394: Changing the Way We Do Multimedia Communications." *IEEE Multimedia*, April-June 2000.

TI90 Texas Instruments Inc. *SN74ACT880 Family Data Manual.* SCSS006C, 1990.

TJAD70 Tjaden, G., and Flynn, M. "Detection and Parallel Execution of Independent Instructions." *IEEE Transactions on Computers,* October 1970.

TOMA93 Tomasevic, M., and Milutinovic, V. *The Cache Coherence Problem in Shared-Memory Multiprocessors: Hardware Solutions.* Los Alamitos, CA: IEEE Computer Society Press, 1993.

TOON81 Toong, H., and Gupta, A. "An Architectural Comparison of Contemporary 16-Bit Microprocessors." *IEEE Micro,* May 1981.

TRIE01 Triebel, W. *Itanium Architecture for Software Developers.* Intel Press, 2001.

TUCK67 Tucker, S. "Microprogram Control for System/360." *IBM Systems Journal,* No. 4, 1967.

TUCK87 Tucker, S. "The IBM 3090 System Design with Emphasis on the Vector Facility." *Proceedings, COMPCON Spring '87,* February 1987.

UNGE02 Ungerer, T.; Rubic, B.; and Silc, J. “Multithreaded Processors.” *The Computer Journal*, No. 3, 2002.

UNGE03 Ungerer, T.; Rubic, B.; and Silc, J. “A Survey of Processors with Explicit Multithreading.” *ACM Computing Surveys*, March, 2003.

VASS03 Vassiliadis,S.; Wong, S.; and Cotofana, S. “Microcode Processing: Positioning and Directions.” *IEEE Micro*, July-August 2003.

VOEL88 Voelker, J. “The PDP-8.” *IEEE Spectrum*, November 1988.

VOGL94 Vogley, B. “800 Megabyte Per Second Systems Via Use of Synchronous DRAM.” *Proceedings, COMPCON ’94*, March 1994.

VONN45 Von Neumann, J. *First Draft of a Report on the EDVAC*. Moore School, University of Pennsylvania, 1945. Reprinted in *IEEE Annals on the History of Computing*, No. 4, 1993.

VRAN80 Vranesic, Z., and Thurber, K. “Teaching Computer Structures.” *Computer*, June 1980.

WALL85 Wallich, P. “Toward Simpler, Faster Computers.” *IEEE Spectrum*, August 1985.

WALL91 Wall, D. “Limits of Instruction-Level Parallelism.” *Proceedings, Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, April 1991.

WANG99 Wang, G., and Tafti, D. “Performance Enhancement on Microprocessors with Hierarchical Memory Systems for Solving Large Sparse Linear Systems.” *International Journal of Supercomputing Applications*, vol. 13, 1999.

WEIN75 Weinberg, G. *An Introduction to General Systems Thinking.* New York: Wiley, 1975.

WEIS84 Weiss, S., and Smith, J. “Instruction Issue Logic in Pipelined Supercomputers.” *IEEE Transactions on Computers*, November 1984.

WEIS94 Weiss, S., and Smith, J. *POWER and PowerPC.* San Francisco: Morgan Kaufmann, 1994.

WEYG01 Weygant, P. *Clusters for High Availability.* Upper Saddle River, NJ: Prentice Hall, 2001.

WHIT97 Whitney, S., et al. “The SGI Origin Software Environment and Application Performance.” *Proceedings, COMPCON Spring ’97*, February 1997.

WICK97 Wickelgren, I. “The Facts About FireWire.” *IEEE Spectrum*, April 1997.

WILK51 Wilkes, M. “The Best Way to Design an Automatic Calculating Machine.” *Proceedings, Manchester University Computer Inaugural Conference*, July 1951.

WILK53 Wilkes, M., and Stringer, J. “Microprogramming and the Design of the Control Circuits in an Electronic Digital Computer.” *Proceedings of the Cambridge Philosophical Society*, April 1953. Reprinted in [SIEW82].

WILK65 Wilkes, M. “Slave memories and dynamic storage allocation,” *IEEE Transactions on Electronic Computers*, April 1965. Reprinted in [HILL00].

WILL90 Williams, F., and Steven, G. “Address and Data Register Separation on the M68000 Family.” *Computer Architecture News*, June 1990.

YEH91 Yeh, T., and Patt, Y. “Two-Level Adaptive Training Branch Prediction.” *Proceedings, 24th Annual International Symposium on Microarchitecture*, 1991.

ZHAN01 Zhang, Z.; Zhu, Z.; and Zhang, X. “Cached DRAM for ILP Processor Memory Access Latency Reduction.” *IEEE Micro*, July-August 2001.

ÍNDICE

C

D

E

F

G

H

I

J

K

L

M

N

O

Ordenación de bits, 405

P

Q

R

S

T

U

V

W

Z

Acrónimos

ACM	Association for Computing Machinery
ALU	Arithmetic and Logic Unit (Unidad Aritmético-Lógica)
ANSI	American National Standards Institute (Instituto nacional americano para normalizaciones)
ASCII	American Standards Code for Information Interchange (Código estándar americano para intercambio de información)
BCD	Binary Coded Decimal (Cifras decimales codificadas en binario)
CD	Compact Disk (Disco compacto)
CD-ROM	Compact Disk-Read Only Memory (Disco compacto de sólo lectura)
CISC	Complex Instruction Set Computer (Computador con repertorio complejo de instrucciones)
CPU	Central Processing Unit (Unidad central de procesamiento)
DMA	Direct Memory Access (Acceso directo a memoria)
DRAM	Dynamic Random-Access Memory (Memoria dinámica de acceso aleatorio)
DVD	Digital Versatile Disk (Disco Versátil Digital)
EEPROM	Electrically Erasable Programmable Read-Only Memory (Memoria de sólo lectura programable eléctricamente borrable)
EPIC	Explicitly Parallel Instruction Computing (Computación con paralelismo de instrucciones explícito)
EPROM	Erasable Programmable Read-Only Memory (Memoria de sólo lectura programable borrable)
HLL	High-Level Language (Lenguaje de alto nivel)
IAR	Instruction Address Register (Registro de dirección de instrucción)
IC	Integrated Circuit (Circuito integrado, chip)
IEEE	Institute of Electrical and Electronics Engineers
ILP	Instruction-Level Parallelism (Paralelismo a nivel de instrucción)
IR	Instruction Register (Registro instrucción)
I/O	Input/Output (Entrada/Salida)
LRU	Least Recently Used (Menos usada recientemente)
LSI	Large-Scale Integration (Integración en gran escala)
MAR	Memory Address Register (Registro de dirección de memoria)
MBR	Memory Buffer Register (Registro intermedio —buffer— de memoria)
MESI	Modify-Exclusive-Shared-Invalid (Modificado-Excluido-No compartido-No válido)
MMU	Memory Management Unit (Unidad de gestión de memoria)
MSI	Medium-Scale Integration (Integración de media escala)
NUMA	Nonuniform Memory Access (Acceso a memoria no uniforme)
OS	Operating System (Sistema Operativo)
PC	Program Counter (Contador de programa)
PCB	Process Control Block (Bloque de control del proceso)
PCI	Peripheral Component Interconnect (Interconexión para componentes periféricos)
PROM	Programmable Read-Only Memory (Memoria sólo lectura programable)
PSW	Processor Status Word (Palabra de estado del procesador)
RAID	Redundant Array of Independent Disks (Array redundante de discos independientes)
RALU	Register/Arithmetic-Logic Unit (Unidad aritmético-lógica con registros)
RAM	Random-Access Memory (Memoria de acceso aleatorio)
RISC	Reduced Instruction Set Computer (Computador con repertorio reducido de instrucciones)
ROM	Read-Only Memory (Memoria de sólo lectura)
SCSI	Small Computer System Interface (Interfaz para computadores pequeños)
SMP	Symmetric Multiprocessors (Multiprocesadores simétricos)
SRAM	Static Random-Access Memory (Memoria de acceso aleatorio estática)
SSI	Small-Scale Integration (Integración en pequeña escala)
ULSI	Ultra Large-Scale Integration (Integración en ultra gran escala)
VLSI	Very Large-Scale Integration (Integración en muy gran escala)
VLIW	Very Long Instruction Word (Palabra de instrucción muy larga)



La finalidad de este texto es que el lector aprenda los principios de diseño e implementación de la organización y arquitectura de los computadores actuales. Por consiguiente, un tratamiento puramente conceptual o teórico sería inadecuado. Este libro utiliza ejemplos de muchas máquinas diferentes para clarificar y reforzar los conceptos presentados. Muchos, pero no todos, de los ejemplos se han ideado a partir de dos familias de computadores: Intel Pentium 4 y PowerPC IBM/Freescale. Estos dos sistemas juntos abarcan la mayoría de las tendencias en diseño de los computadores de hoy en día. El Pentium 4 es esencialmente un computador con un conjunto complejo de instrucciones (CISC, Complex Instruction Set Computer) con características RISC, mientras que el PowerPC es esencialmente un computador con un conjunto reducido de instrucciones (RISC, Reduced Instruction Set Computer). Ambos sistemas utilizan principios de diseño superescalar y ambos soportan configuraciones multiprocesador.

Incluye:



LibroSite es una página web asociada al libro, con una gran variedad de recursos y material adicional tanto para los profesores como para estudiantes. Apoyos a la docencia, ejercicios de autocontrol, enlaces relacionados, material de investigación, etc., hacen de LibroSite el complemento académico perfecto para este libro.

Otro libro de interés:



Tanenbaum, Andrew S.
Organización de Computadoras, 4ª. Edición. Madrid, Pearson Prentice Hall, 2000. ISBN 9701703995



www.pearsoneducacion.com

